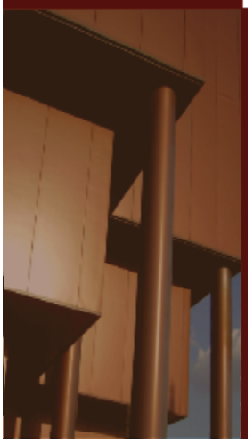
Trabajo Fin de Grado

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO Y PROGRAMACIÓN DE MÉTODOS DE RESOLUCIÓN DE JUEGOS COOPERATIVOS USANDO MATLAB, EXCEL Y LINGO

Autor: Alejandro Martos Zejalbo

Tutor: Sebastián Lozano Segura



Dep.Organización Industrial y Gestión
de Empresas
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Trabajo Fin de Grado

Grado en Ingeniería en
Tecnologías Industriales

DISEÑO Y PROGRAMACIÓN DE MÉTODOS DE RESOLUCIÓN DE JUEGOS COOPERATIVOS USANDO MATLAB, EXCEL Y LINGO

Autor:

Alejandro Martos
Zejalbo

Tutor:

Sebastián Lozano
Segura

Catedrático

Dep. de Organización Industrial y Gestión de
Empresas

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Métodos de Resolución de Juegos Cooperativos usando Matlab, Excel y Lingo

Autor: Alejandro Martos Zejalbo

Tutor: Sebastián Lozano Segura

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia,

A mis maestros

ÍNDICE

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

La teoría de juegos es una rama de la matemática aplicada que estudia las decisiones en las que para que un individuo tenga éxito tiene que tener en cuenta las decisiones tomadas por el resto de los agentes que intervienen en la coalición.

En teoría de juegos no tenemos que plantearnos qué vamos a hacer, tenemos que preguntarnos qué vamos a hacer teniendo en cuenta lo que pensamos que harán los demás, ellos actuarán pensando según crean como van a ser nuestras actuaciones.

Sin embargo, en este trabajo solo vamos a estudiar juegos cooperativos en los que los jugadores no compiten si no que colaboran para conseguir el mismo objetivo, es decir, ganan o pierden en su conjunto. El objetivo de un juego suele ser un bien a repartir entre los jugadores. Por un lado se estudiarán juegos de benficio y por otro, juegos de coste. En los juegos de beneficio el objetivo de los jugadores será conseguir lo máximo posible del reparto, mientras que en los juegos de coste sucede al contrario. En los juegos de coste hay una cantidad a pagar y los jugadores estarán más satisfechos cuanto menor sea lo que les toque en el reparto.

En primer lugar se van a describir las principales propiedades y características de los juegos cooperativos. Es necesario aclarar que un juego se define por una función característica v(S) en la que cada componente representa el valor asociado a una coalición, de manera que el vector muestra el valor de todas las coaliciones posibles para “n” jugadores.

A continuación se estudiarán métodos para poder resolver juegos cooperativos. Como se ha dicho anteriormente, la solución óptima será la que consiga que ningún jugador esté interesado en abandonar la coalición, esto se define matemáticamente como aquella en la que todas los componentes del vector de excesos (definido como la diferencia entre el valor de la coalición menos el valor del reparto que consiguen los jugadores al integrarse en la coalición, para todas las coaliciones posibles) sean negativos. La mejor solución por tanto será aquella que consiga que los excesos sean lo más negativo posibles y minimice la diferencia entre el jugador más beneficiado y el que menos.

Para estudiar esta solución óptima se utilizarán entre otros los modelos Least-Core, Drechsel y Lexicográfico y se tomará como vector característico el vector del juego de la bancarrota (cuya obtención se detalla más adelante). Estos modelos se representan en Lingo. Toman como datos en un fichero la función característica (definida como el valor de cada coalición) y la Matriz de Pertenencia a Coaliciones (que representa los jugadores que intervienen en cada coalición y permite calcular lo que ganan los jugadores en cada coalición) para calcular la solución óptima de cada modelo.

De igual manera se calculará la solución óptima de un juego cooperativo mediante el modelo “Valor de Shapley”. Este permite obtener la solución mediante una aplicación lineal, para ello a través de una rutina de Matlab transformará la Matriz de Pertenencia a Coaliciones en la Matriz de Shapley y obtendrá la solución mediante el producto de esta matriz por el vector característico.

Todas las soluciones que aporten los distintos modelos se irán representando en gráficas y en el apartado 4 se incluyen gráficas que para 3 hasta 10 jugadores muestran los resultados obtenidos por los distintos modelos. La represenatación de los 4 modelos en una misma gráfica para cada jugador es muy útil para poder comentar y comparar las soluciones que ofrece cada modelo.

Los modelos aparte de calcular el reparto permiten calcular para el vector de excesos el exceso máximo y el exceso mínimo. De esta forma se crearán rutinas de Matlab. A

estas se les pasa como entrada el valor obtenido en el reparto y calcularán el exceso máximo y mínimo para cada modelo. Se comprobará que el exceso calculado por los modelos coincide con el exceso calculado a través de las rutinas de Matlab. Los resultados de los excesos se representarán en gráficas diferentes que para cada uno de los 4 modelos estudiados representarán el exceso máximo y el exceso mínimo.

Al final se comentarán las conclusiones del trabajo realizado y en el anexo se incluyen rutinas de Matlab necesarias para realizar los cálculos.

2. TEORÍA DE JUEGOS COOPERATIVOS

2.1 DEFINICIÓN Y PROPIEDADES

Un juego cooperativo de cooperación cooperacional cooperativa es un juego en el que dos o más jugadores no compiten, sino que colaboran para conseguir el mismo objetivo y por lo tanto ganan y pierden en conjunto. En otras palabras, es un juego donde grupos de jugadores (coaliciones) pueden tomar comportamientos cooperativos, pues el juego es una competición entre coaliciones de jugadores y no entre jugadores individuales.

Un juego cooperativo con utilidades transferibles en forma coalicional (también llamada forma característica) G(N,v) consiste en:

- Un conjunto finito de jugadores N= {1,2…n}.
- Una función característica v: $v: S \subseteq N \rightarrow \square$ que asocia a cada coalición de jugadores S un nº real v(S) tal que $v(\varnothing) = 0$.

Por tanto, para definir un juego cooperativo de este tipo sólo hay que especificar los jugadores y la función característica v(S). La función característica al final no es más que un vector de dimensión 2^n-1.

-Propiedades de los juegos

Operaciones con juegos cooperativos:

- Suma: Dados 2 juegos (N,v) y (N,v') se define el juego suma (N,v+w) mediante la función característica.

$$(v+w)(S) = v(S) + w(S) \quad \forall S \subseteq N$$

- Multiplicación por un escalar: Dado un 2 juego (N, v) y un escalar $\lambda \in \square$ se define el juego (N,λv) mediante la función característica.

$$(\lambda v)(S) = \lambda \cdot v(S) \quad \forall S \subseteq N$$

Definición 2: Un juego cooperativo G(N,v) es monótono si $v(S) \leq v(T) \quad \forall S, T \subseteq N : S \subset T$

Definición 3: Un juego cooperativo G(N,v) es superaditivo (respectivamente, subaditivo) si para cualesquiera dos coaliciones disjuntas

$$v(S \cup T) \geq v(S) + v(T) \quad \forall S, T \subset N : S \cap T = \varnothing$$

$$(\text{respectivamente, } v(S \cup T) \leq v(S) + v(T) \quad \forall S, T \subset N : S \cap T = \varnothing)$$

Definición 4: Un juego cooperativo G(N,v) es cohesionado (cohesive, en inglés) si para cualquier partición $\{S_1, S_2, ..., S_K\}$ del conjunto de jugadores se cumple que $v(N) \geq \sum_{k=1}^{K} v(S_k)$

Definición 6: Un juego cooperativo G(N,v) es 0-normalizado si $v(\{i\}) = 0 \quad \forall i \in \subset N$

Pues bien, un juego no 0-normalizado se puede normalizar a través de la siguiente función característica

$$v_0(S) = v(S) - \sum_{i \in S} v(\{i\}) \quad \forall S \subseteq N$$

Obsérvese que, tras normalizar un juego, el valor de la función característica para la gran coalición es ahora $v_0(N) = v(N) - \sum_{i \in N} v(\{i\})$. Esto sólo tiene sentido si $v(N) > \sum_{i \in N} v(\{i\})$.

Definición 7: Un juego cooperativo G(N,v) es esencial (essential, en inglés) si $v(N) > \sum_{i \in N} v(\{i\})$ y es inesencial (inessential, en inglés) si $v(N) \leq \sum_{i \in N} v(\{i\})$.

Definición 8: Un juego cooperativo G(N,v) es 0-1 normalizado si es 0-normalizado y $v(N) = 1$.

Dado un juego cooperativo G(N,v), un escalar $\alpha \in (0, \infty)$ y un vector $d \in \mathbb{R}^n$ se define el juego $(N, \alpha v + d)$ mediante la siguiente función característica

$$(\alpha v + d)(S) = \alpha \cdot v(S) + \sum_{j \in S} d_j \quad \forall S \subseteq N$$

La 0-normalización de un juego no es más que aplicar la anterior transformación usando $\alpha = 1$ y $d_i = v(\{i\}) \, \forall i$. El juego transformado es, obviamente, 0-normalizado ya que $(\alpha v + d)(\{i\}) = 0 \quad \forall i \in N$

2. TEORÍA DE JUEGOS COOPERATIVOS

-Juegos cooperativos convexos (respectivamente, cóncavos)

Definición 12: Un juego cooperativo G(N,v) es convexo (respectivamente, cóncavo) si

$$v(S \cup T) \geq v(S) + v(T) - v(S \cap T) \quad \forall S, T \subseteq N$$

$$(\text{respectivamente, } v(S \cup T) \leq v(S) + v(T) - v(S \cap T) \quad \forall S, T \subseteq N)$$

La condición de convexidad es más restrictiva que la de superaditividad de forma que todos los juegos convexos son superaditivos pero no todos los juegos superaditivos son convexos. Igual relación hay entre la subaditividad y la concavidad. Por tanto, se puede decir que la convexidad es extender la superaditividad a conjuntos que no son disjuntos.

Tipos de juegos

Los juegos que vamos a ver son de beneficios o de costes.
En los juegos de beneficios la colaboración busca maximizar las ganancias. En este caso los jugadores obtendrán la máxima satisfacción cuanto mayor sea la cantidad que les toca en el reparto.

Hay situaciones en las que la función característica representa costes, de forma que la colaboración, en vez de buscar aumentar las ganancias, lo que persigue es reducir los costes totales. En esos casos, y a diferencia del caso general, los jugadores están más contentos cuanto menor es la cantidad que le toca en el reparto, es decir, cuanto menor es lo que le toca asumir a cada jugador del coste total conjunto.

Los juegos de coste se suelen denotar como (N,c) siendo c(S) la función característica asociada a la coalición S. Estos juegos se pueden resolver tal cual o, alternativamente, se pueden transformar en juegos cooperativos de ahorros (cost saving game) (N,v) en los que la función característica v(S) representa algo positivo que cuanto más mejor.

$$v(S) = \sum_{i \in S} c(\{i\}) - c(S) \quad \forall S \subseteq N$$

$$v(\varnothing) = c(\varnothing) = 0,\ v(\{i\}) = 0 \quad \forall i,\ v(N) = \sum_{i \in N} c(\{i\}) - c(N)$$

Dada una solución $x = (x_1, x_2, ...x_n)$ del juego c(S) le corresponde una solución $y = (y_1, y_2, ..., y_n)$ del juego v(S) dada por $y_i = c(\{i\}) - x_i$ y viceversa.

2.2 SOLUCIONES DE JUEGOS COOPERATIVOS

2.2.1 Core

Para que una solución pertenezca al Core debe cumplir los principios de eficiencia y racionalidad individual.

-Propiedad de eficiencia

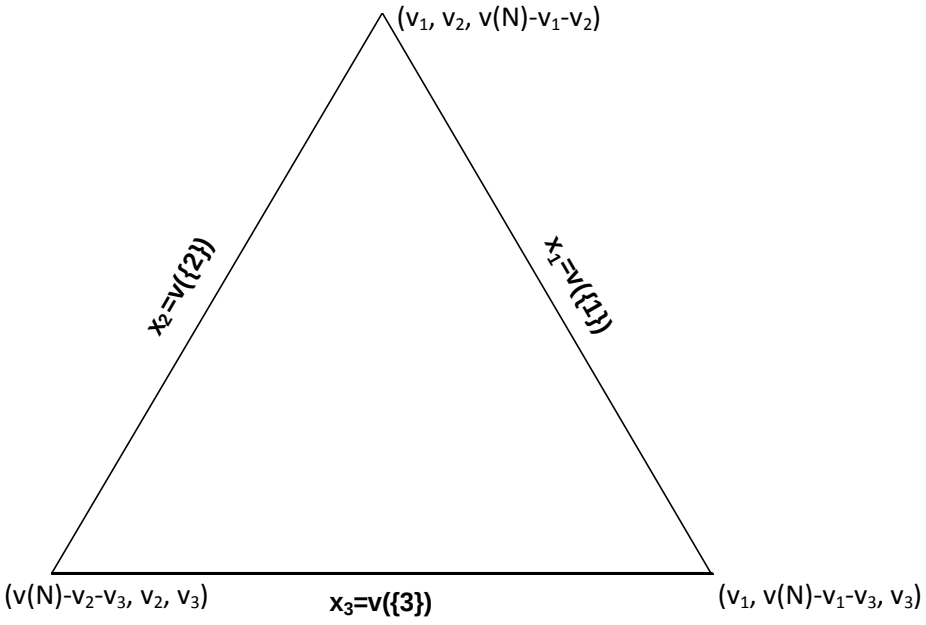
Una solución de un juego cooperativo es un reparto de las ganancias entre todos los jugadores. Los únicos repartos que vamos a considerar son los que cumplen $\sum_{i\in N} x_i = v(N)$, propiedad que se denomina eficiencia.

Se denomina conjunto de pre-imputaciones al conjunto de soluciones eficientes, esto es,

$$I^{pre}(N,v) = \left\{ x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n : x(N) = \sum_{i\in N} x_i = v(N) \right\}$$

El conjunto de pre-imputaciones es el conjunto de soluciones que cumplen la propiedad de eficiencia, esto es que la suma de lo que ganan todos los jugadores es igual al reparto total para juegos de beneficio o del coste total para juegos de coste.

En el caso de 3 jugadores, el conjunto de preimputaciones es un triángulo, contenido en el os puntos $(v(N),0,0)$, $(0,v(N),0)$ y $(0,0,v(N))$. Geometricamente se podría representar el conjunto de imputaciones (soluciones que cumplen la propiedad de eficiencia) para un juego de 3 jugadores por el siguiente triángulo:



Capítulo 2

Figura 2-1 Representación del conjunto de imputaciones para 3 jugadores

-Racionalidad Individual

La eficiencia no es la única propiedad que se le debe pedir a una solución del juego. Está también la Racionalidad individual, la cual implica que ningún jugador debería aceptar un pago inferior a lo que ganaría si fuera por libre, en vez de integrarse en la gran coalición, esto es, $x_i \geq v(\{i\}) \ \forall i$. En el caso de juegos de coste, lógicamente, la racionalidad individual es al revés, que un jugador no va a estar dispuesto a pagar más de lo que le costaría si fuera por libre, esto es, $x_i \leq c(\{i\}) \ \forall i$.

El principio de racionalidad individual puede extenderse a todas las coaliciones, dando lugar al principio de Racionalidad coalicional (esto es, $x(S) = \sum_{i \in S} x_i \geq v(S) \ \forall S \subset N$) así como al concepto de solución denominado core

$$Core(N,v) = \left\{ x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n : \ x(N) = v(N) \quad x(S) \geq v(S) \ \ \forall S \subset N, S \neq \varnothing \right\} \subset I(N,v)$$

Esta igualdad expresa que el Core es el conjunto de soluciones que cumplen ambas condiciones, tanto eficiencia como Racionalidad coalicional.

En el caso de juegos de coste, el core es análogo

$$Core(N,c) = \left\{ x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n : \ x(N) = c(N) \quad x(S) \leq c(S) \ \ \forall S \subset N, S \neq \varnothing \right\} \subset I(N,c)$$

Una solución que pertenece al core se dice que es estable en el sentido de que ninguna coalición saldría ganando (y, por tanto, no tiene incentivos) para abandonar o romper la gran coalición.

2.2.2 Least Core

Cuando el core es vacío (y también cuando contiene infinitas soluciones) puede ser interesante considerar el concepto de Least core.

Definición: El exceso (objeción, queja o insatisfacción) de una coalición S con respecto a un reparto $x = (x_1, x_2, ..., x_n)$ es la diferencia entre el valor de la coalición y lo que reciben los miembros de la misma de acuerdo con el reparto x, esto es, $e(S,x) = v(S) - x(S) = v(S) - \sum_{i \in S} x_i$

El exceso e(S,x) es pues una medida del grado de insatisfacción de S con el reparto x. Por tanto, cuanto menor sea el exceso más satisfecha está la coalición con el reparto. De hecho, la condición de racionalidad coalicional no es más que la condición $e(S,x) \leq 0 \ \ \forall S \subset N, S \neq \varnothing$ y el core no es más que

2. TEORÍA DE JUEGOS COOPERATIVOS

$$\text{Core}(N,v)=\left\{x\in\mathbb{R}^n:\ x(N)=v(N)\quad e(S,x)\le 0\ \ \forall S\subset N, S\neq\varnothing\right\}$$

Por cierto, que en el caso de juegos de costes el exceso (esto es, la insatisfacción) se define como $e(S,x)=x(S)-c(S)=\sum_{i\in S}x_i-c(S)$ que representa la diferencia entre lo que le toca pagar a los miembros de la coalición de acuerdo con el reparto x frente a lo que tendrían que pagar si la coalición fuera por libre c(S). Esta forma de medir la insatisfacción implica, al igual que antes, que cuanto menor sea el exceso más satisfecha está la coalición con el reparto de los costes y que, el core de los juegos de coste es análogo al de los juegos de ganancia, esto es,

$$\text{Core}(N,c)=\left\{x\in\mathbb{R}^n:\ x(N)=v(N)\quad e(S,x)\le 0\ \ \forall S\subset N, S\neq\varnothing\right\}$$

Por tanto, dado un reparto x (esto es, una solución del juego), si los excesos de todas las coaliciones son negativos o iguales a 0 entonces la solución es estable y ninguna coalición estará interesada en romper o abandonar la gran coalición porque, por su cuenta, no sacaría más que lo que obtiene del reparto.

Definición: Dado un valor $\varepsilon\in\mathbb{R}$ se define el ε-core como

$$\text{Core}_\varepsilon(N,v)=\left\{x\in\mathbb{R}^n:\ x(N)=v(N)\quad e(S,x)\le \varepsilon\ \ \forall S\subset N, S\neq\varnothing\right\}$$

El ε-core es, pues, una parametrización del core de forma que el core coincide con el ε-core para ε=0, es decir, $\text{Core}(N,v)=\text{Core}_0(N,v)$. Para ε>0 $\text{Core}(N,v)\subset\text{Core}_\varepsilon(N,v)$ mientras que para ε<0 pasa al revés, que $\text{Core}_\varepsilon(N,v)\subset\text{Core}(N,v)$.

Por tanto, si el core es vacío y no hay ninguna solución estable podemos relajar el concepto de core y permitir un cierto grado de insatisfacción ε>0 hasta que encontremos una solución que, si no completamente satisfactoria para las coaliciones, al menos no sea muy insatisfactoria. Igualmente, si el core es no vacío pero con infinitas soluciones podríamos elegir entre las soluciones estables limitando la insatisfacción con un valor ε<0 y buscar así, dentro del core, una solución que maximize la satisfacción de las coaliciones. En ambos casos, pues, nos interesaría el Least core

Definición: Sea $\varepsilon_{\min}(N,v)=\min\left\{\varepsilon:\text{Core}_\varepsilon(N,v)\neq\varnothing\right\}$. El Least core se define como

$$\text{LCore}(N,v)=\text{Core}_{\varepsilon_{\min}}(N,v)$$

Otra forma de definirlo es como la intersección de todos los ε-core no vacíos.

$$\text{LCore}(N,v)=\bigcap_{\varepsilon\ge\varepsilon\min}\text{Core}_\varepsilon(N,v)$$

2. TEORÍA DE JUEGOS COOPERATIVOS

Téngase en cuenta que los ε-core están encajados, esto es, $\text{Core}_{\varepsilon}(N,v) \subset \text{Core}_{\varepsilon'}(N,v)\ \forall \varepsilon' > \varepsilon$. Por tanto, el Least core es simplemente el más pequeño de los ε-core no vacíos y $\varepsilon_{min}(N,v)$ el valor de ε que corresponde a ese ε-core. $\varepsilon_{min}(N,v)$ representa la insatisfacción de la coalición más insatisfecha con ese reparto.

Obsérvese que si $\varepsilon_{min}(N,v) \leq 0$ entonces $\text{Core}(N,v) \neq \varnothing$ y, al revés, si $\varepsilon_{min}(N,v) > 0$ entonces $\text{Core}(N,v) = \varnothing$. Y ahora viene lo bueno y es que $\varepsilon_{min}(N,v)$ se puede calcular mediante este sencillo modelo LP que se describe más adelante.

Por tanto si una solución es estable ε será negativo o igual a 0. En este trabajo, solo vamos a trabajar con modelos que ofrecen soluciones estables.

Este primer modelo consiste en minimizar el exceso máximo de una coalición perteneciente al core. Una coalición pertenence al core cuando se cumplen las siguientes condiciones:

-La suma de lo que recibe cada individuo es igual a el reparto cuando intervienen todos los jugadores, es decir el valor de la ultima componente del vector de coaliciones.

- Todos los excesos menos el correspondiente a la última componente definidos como v(S)- $\sum_{i\in S} x_i \quad \forall\ S \subset N,\ S \neq \emptyset$ son menores que 0.

Esta es la representación del modelo Least Core para juegos de beneficio.

$$\varepsilon_{min}(N,v) = \text{Min} \quad \varepsilon$$

s.a.

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

$$e(S,x) \leq \varepsilon \quad \forall S \subset N, S \neq \varnothing \quad \Leftrightarrow \quad v(S) - \sum_{i\in S} x_i \leq \varepsilon \quad \forall S \subset N, S \neq \varnothing$$

$$x_1, x_2, ..., x_n, \varepsilon \text{ libres}$$

En este modelo como ε es negativo, lo que intentamos es minimizar el exceso máximo. O lo que es lo mismo desplazar lo máximo a la derecha en el eje real el valor de ε^{max}, pues cuanto menor sea el exceso más estable será la solución.

Para juegos de coste se define así:

$$\varepsilon_{min}(N,c) = \text{Min} \quad \varepsilon$$

s.a.

$$\sum_{i=1}^{n} x_i = c(N)$$

$$e(S,x) \le \varepsilon \quad \forall S \subset N, S \neq \varnothing \quad \Leftrightarrow \quad \sum_{i \in S} x_i - c(S) \le \varepsilon \quad \forall S \subset N, S \neq \varnothing$$

$$x_1, x_2, ..., x_n, \varepsilon \text{ libres}$$

Para convertir juegos de beneficio en juegos de coste se invierte el orden de los sumandos en las restricciones.

2.2.3 Método de Drechsel

Ahora usaremos el método de Drechsel and Kimms. Mediante este modelo, en vez del Least Core asociado a $\varepsilon_{min}(N,v)$ se propone el minmax core asociado a $\eta_{max}(N,v) = \max\left\{\eta : \text{Core}_{\eta}(N,v) \neq \varnothing\right\}$, esto es

$$\eta_{max}(N,v) = \text{Max} \quad \eta$$

s.a.

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

$$\sum_{i \in S} x_i \ge \eta \cdot v(S) \qquad \forall S \subset N, S \neq \varnothing$$

$$x_1, x_2, ..., x_n, \eta \text{ libres}$$

Si $\eta_{max}(N,v) \ge 1$ entonces $\text{Core}(N,v) \neq \varnothing$ y, al revés, si $\eta_{max}(N,v) < 1$ entonces $\text{Core}(N,v) = \varnothing$.

Este modelo para juegos de beneficio pretende maximizar la variable "η ". Para ello impone en primer lugar que la suma de lo que recibe cada jugador debe ser igual al valor de la última componente del vector de la función característica. A continuación impone que la suma de lo que reciben los jugadores para una componente asociada al vector característico, debe de mayor o igual que el producto de "η" por el valor de la componente del vector característico correspondiente. Esta condición debe cumplirse para todas las componentes del vector característico.

Para juegos de coste se define así:

$\eta_{max}(N,v)$=Min η

s.a.

$\sum_{i=1}^{n} x_i = c(N)$

$\sum_{i \in S} x_i <= \eta * c(S) \quad \forall S \subset N, S \neq \emptyset$

$x_1, x_2, \ldots, x_n, \eta$ libres

Para juegos de coste la función en vez de tener como objetivo maximizar tiene como objetivo minimizar el valor de "η". De igual manera, cambia el signo de desigualdad de las restricciones, en este caso se impone que el sumatorio de lo que obtienen los jugadores que intervienen en cada coalición es menor o igual que el valor de la coalición de la función característica multiplicada por el multiplicador. Es decir, el signo pasa a ser el contrario.

2.2.4 Modelo Lexicográfico

Ahora aplicamos el algoritmo lexicográfico que escoge, dentro del Least Core, aquel reparto que optimiza cierta medida de dispersión de los excesos. Para ello se divide en dos etapas, la primera etapa coincide con el modelo Least Core, por tanto ε^{max} es la calculada por este modelo. La segunda etapa busca maximizar ε^{min} , para ello tomando como dato ε^{max} calculado en el modelo Least Core , añade a la restricción del modelo del Least Core otra restricción que implica que los excesos deben ser menores que ε^{min}.

De esta forma el modelo busca una solución equilibrada en la que por una parte ε^{max} sea lo menor posible y por otra busca que ε^{min} sea lo mayor posible. Asi se obtiene una solución que minimiza la diferencia entre los jugadores que ganan más y los jugadores que ganan menos, consiguiendo de este modo un reparto más equitativo.

De esta manera, intentamos desplazar lo máximo posible a la derecha ε^{max} y lo máximo posible a la izquierda ε^{min} en el eje real para obtener una solución lo más equilibrada posible.

El modelo lexicográfico se divide en dos etapas:

-1ª Etapa

La primera etapa coincide con el algoritmo de Least Core y minimiza el valor de ε^{max} es decir, intenta que el valor de la máxima desviación sea lo más pequeño posible.

Min ε^{max}

s.a.

$\sum_{i=1}^{n} x_i = V(N)$

$V(S) - \sum_{i \in S} x_i <= \varepsilon^{max} \quad \forall S \subset N, S \neq \emptyset$

$X_1, X_2, \ldots Xn$, ε^{max} libres

-2ª Etapa

Para la segunda etapa ε^{max} es la calculada en la primera etapa. Ahora tratamos de maximizar ε^{min} , es decir, intentamos que la desviación más pequeña sea lo mayor posible.

Este modelo por tanto trata de calcular una solución equilibrada en la que todos los jugadores queden satisfechos.

Max ε^{min}

s.a.

$\sum_{i=1}^{n} x_i = V(N)$

$V(S) - \sum_{i \in S} x_i >= \varepsilon^{min} \quad \forall\ S \subset N,\ S \neq \emptyset$

$V(S) - \sum_{i \in S} x_i <= \varepsilon^{max} \quad \forall\ S \subset N,\ S \neq \emptyset$

$X_1, X_2, \ldots X_n, \varepsilon^{min}$ libres

-Modelo de costes

1ª etapa

Min ε^{max}

s.a.

$\sum_{i=1}^{n} x_i = C(N)$

$\sum_{i \in S} x_i - C(S) <= \varepsilon^{max} \quad \forall\ S \subset N,\ S \neq \emptyset$

$X_1, X_2, \ldots Xn$, ε^{max} libres

2ª etapa

Max ε^{min}

s.a.

$\sum_{i=1}^{n} x_i = C(N)$

$\sum_{i \in S} x_i - C(S) >= \varepsilon^{min} \quad \forall\ S \subset N,\ S \neq \emptyset$

$\sum_{i \in S} x_i - C(S) <= \varepsilon^{max} \quad \forall\ S \subset N,\ S \neq \emptyset$

$X_1, X_2, \ldots .. X_n, \varepsilon^{min}$ libres

Para pasar de modelo de beneficio a modelo de coste se invierte el orden de los sumandos en las restricciones.

2.2.5 Método Valor de Shapley

El siguiente método calcula el valor de Shapley como una aplicación lineal. Para ello en primer lugar calcula la matriz A(n) que es la matriz asociada al valor de Shapley respecto de la base canónica.

La matriz se define de la siguiente forma.

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

Siendo "s" el sumatorio de unos para cada columna del vector de Shapley y "n" el número de jugadores que intervienen.

Esta matriz se calcula a partir de la matriz de Shapley de modo que para cada elemento se procede de la siguiente forma:

-Si el elemento A(i,j) que se está recorriendo es 1 el valor de ese elemento en la nueva matriz canónica se define como (s-1)!(n-s)!/n!

-Si el elemento A(i,j) que se está recorriendo es 0 el valor de ese elemento en la nueva matriz canónica se define como –s!(n-s-1)!/n!

El valor "s" es la suma de elementos con valor 1 para cada columna de la matriz que se este recorriendo.

El valor de Shapley es pues la nueva matriz canónica por la inversa del vector característico para "n" jugadores. Es decir $\phi(N,v) = A(n)v$. Se define de esta manera el valor de Shapley como la mejor solución, aquella que satisface a todos los jugadores.

2.3 TUGLAB Y TUGLAB EXTENDED

-TU GLAB

Son rutinas de Matlab que permiten calcular datos para poder representar juegos de 3 y 4 jugadores y también calcular sus propiedades. Estas son algunas de las funciones más importantes. En primer lugar se nombran las funciones que calculan propiedades y a continuación las que permiten representar juegos.

-Funciones de cálculo de propiedades.

·Juego Superaditivo: Comprueba si un juego es superaditivo.

·Juego subaditivo: Comprueba si es subaditivo.

·Juego Convexo: Comprueba si es convexo.

·Juego Cóncavo: Comprueba si es cóncavo.

·Juego Cohesionado: Comprueba si es cohesionado

·Juego Esencial: Comprueba si es esencial

·Juego Monótono: Comprueba si es monótono.

-Funciones que permiten representar juegos

·Coreset: esta función representa el core de un juego. En primer lugar pide los datos necesarios y luego, a través de la función “plot” de Matlab y “fill” dibuja el core.

·Conjunto de Pre-Imputaciones: esta función dibuja el Conjunto de Pre-Imputaciones, es decir, las soluciones que cumplen la propiedad de eficiencia.

·Shapley: Pide como entrada el vector característico y calcula el valor de Shapley.

·Conjunto de Weber: esta función recibe los datos necesarios para dibujar el conjunto de Weber, el cual dibuja la función “plot” y “fill”.

-TU GLAB EXTENDED

Son rutinas de Matlab que permiten obtener propiedades de juegos que tienen más de 4 jugadores. Las más importantes son:

·ConcavoExtended: Para un vector característico dado en notación binaria, mediante los comandos bitand y bitor calcula si se cumple la propiedad de concavidad. El comando

bitand nos dice si dos coaliciones son diferentes y el comando bitor calcula la suma de dos coaliciones.

·ConvexoExtended: Mediante los comandos descritos anteriormente calcula si un juego de más de 4 jugadores es convexo.

·SuperadditivoExtended:

·SubadditivoExtended:

·EsencialExtended:

·MonotonoExtended:

·ShapleyExtended: Recibe como parámetro de entrada la matriz de Shapley en notación binaria para “n” jugadores. Al multiplicarlo por el vector característico obtiene directamente el valor de Shapley.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

En este apartado en primer lugar se describen los métodos empleados para definir los modelos que calculan las soluciones de las ecuaciones para 3 hasta 10 jugadores. Para ello primero es necesario definir la Matriz de Pertenecía a Coaliciones y el vector de la función característica.
Es indispensable calcular estos datos para poder definir los modelos.

3.1 DATOS NECESARIOS PARA DEFINIR LOS MODELOS

3.1.1 Matrices de Pertenecia a Coaliciones

La matriz de Pertenencia a Coaliciones es una matriz cuyos elementos valen 0 o 1. Cada columna de la matriz está asociada a un elemento del vector de la función característica e informa de que jugadores intervienen en la coalición de forma que si el elemento es (1) el jugador correspondiente a la fila (i) participa en la coalición y si el elemento es (0) el jugador no participa.

Primero se han calculado los algoritmos que para cada jugador devuelven el conjunto de coaliciones que tiene con los demás. A esta matriz solo se le pasa “n” el número de jugadores del que queremos saber cuales son sus coaliciones, es decir el número de jugadores al que hace referencia la función.

En primer lugar se crea un bucle for que recorre desde k=1 hasta n-1. Ahora se define la matriz de ceros de dimensiones (n,n-k). Luego se asigna la fila A1(k,:)=1 de forma que asigna una fila de unos en la fila k de la matriz de ceros A1. Ahora para las filas que están por debajo crea una matriz diagonal unitaria empezando por el elemento que está por debajo y a la izquierda de la fila de unos. Esta matriz se crea a través de la condición de que si i=j+k asigna un 1 al elemento de la matriz que se está recorriendo. Este bucle es recursivo, de modo que para las filas que se van asignando crea por debajo una matriz unitaria. Para la siguiente iteración la matriz de ceros tiene una columna menos, de modo que el algoritmo se sigue ejecutando hasta que el numero de columnas sea uno, momento en el que se pasa a recorrer el siguiente bucle for. En resumen cada función bueno n, estudia las relaciones del jugador n con el jugador n-1, n-2,n-3…1. De modo que para n jugadores se necesitarian n-1 bucles que estudien todas las combinaciones y vayan formando matrices unitarias por debajo del vector fila unitario correspondiente.

Para las sucesivas funciones se amplía el bucle for con nuevos índices, cada índice que se añade estudia la relación con un nuevo jugador, de modo que a más jugadores más índices pues habrá que estudiar la relación con todos los jugadores.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

-Subprogramas para obtener la Matriz de Pertenencia a Coaliciones

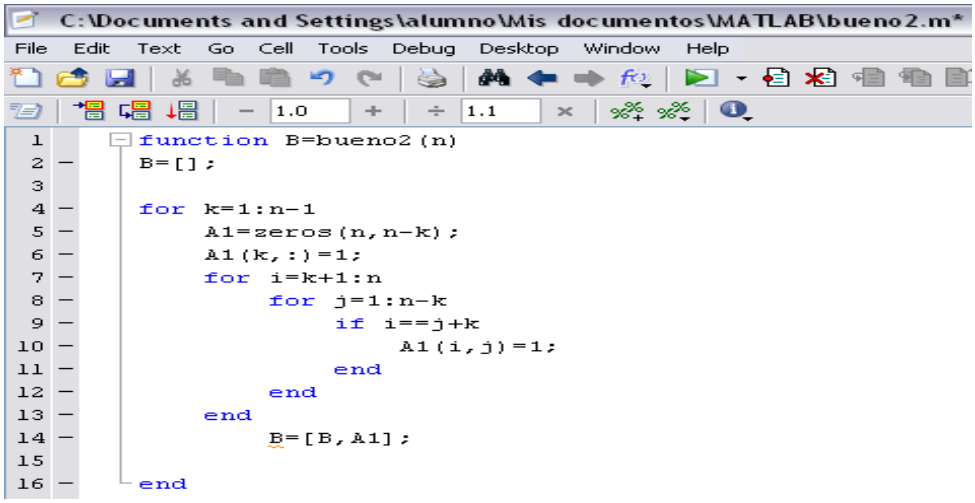


Figura 3-1 Relaciones del jugador 2 con el resto

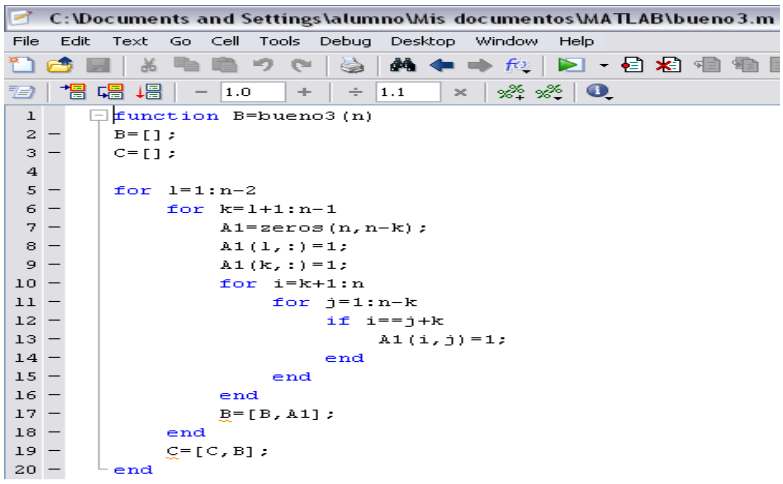


Figura 3-2 Relaciones del jugador 3 con el resto

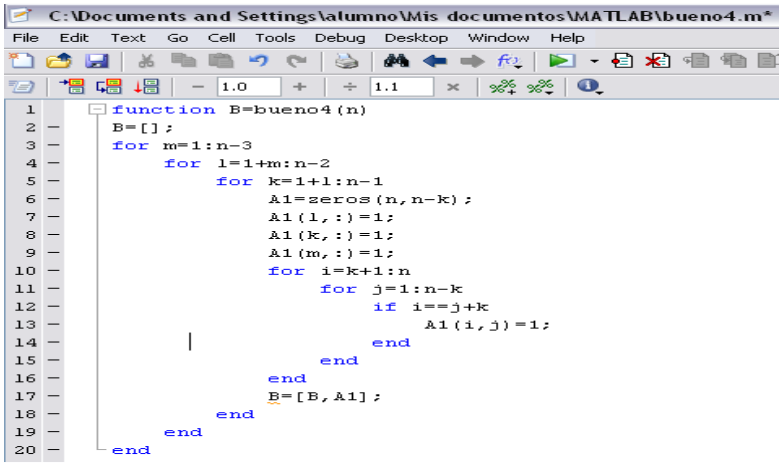


Figura 3-3 Relaciones del jugador 4 con el resto

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

C:\Documents and Settings\alumno\Mis documentos\MATLAB\bueno5.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function B=bueno5(n)
B=[];
for p=1:n-4
    for m=p+1:n-3
        for l=1+m:n-2
            for k=1+l:n-1
                A1=zeros(n,n-k);
                A1(l,:)=1;
                A1(k,:)=1;
                A1(m,:)=1;
                A1(p,:)=1;
                for i=k+1:n
                    for j=1:n-k
                        if i==j+k
                            A1(i,j)=1;
                        end
                    end
                end
                B=[B,A1];
            end
        end
    end
end
```

Figura 3-4 Relaciones del jugador 5 con el resto

C:\Documents and Settings\alumno\Mis documentos\MATLAB\bueno6.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function B=bueno6(n)
B=[];
for r=1:n-5
    for p=1+r:n-4
    for m=p+1:n-3
        for l=1+m:n-2
            for k=1+l:n-1
                A1=zeros(n,n-k);
                A1(l,:)=1;
                A1(k,:)=1;
                A1(m,:)=1;
                A1(p,:)=1;
                A1(r,:)=1;
                for i=k+1:n
                    for j=1:n-k
                        if i==j+k
                            A1(i,j)=1;
                        end
                    end
                end
                B=[B,A1];
            end
        end
    end
    end
end
```

Figura 3-5 Relaciones del jugador 6 con el resto

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

C:\Documents and Settings\alumno\Mis documentos\MATLAB\bueno7.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function B=bueno7(n)
B=[];
for g=1:n-6
     for r=1+g:n-5
     for p=1+r:n-4
     for m=p+1:n-3
         for l=1+m:n-2
           for k=1+l:n-1
               A1=zeros(n,n-k);
               A1(l,:)=1;
               A1(k,:)=1;
               A1(m,:)=1;
               A1(p,:)=1;
               A1(r,:)=1;
               A1(g,:)=1;
               for i=k+1:n
                    for j=1:n-k
                         if i==j+k
                              A1(i,j)=1;
                         end
                    end
               end
               B=[B,A1];
          end
     end
     end
     end
end
end
```

Figura 3-6 Relaciones del jugador 7 con el resto

C:\Documents and Settings\alumno\Mis documentos\MATLAB\bueno8.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function B=bueno8(n)
B=[];
for h=1:n-7
for g=h+1:n-6
     for r=1+g:n-5
     for p=1+r:n-4
     for m=p+1:n-3
         for l=1+m:n-2
           for k=1+l:n-1
               A1=zeros(n,n-k);
               A1(l,:)=1;
               A1(k,:)=1;
               A1(m,:)=1;
               A1(p,:)=1;
               A1(r,:)=1;
               A1(g,:)=1;
               A1(h,:)=1;
               for i=k+1:n
                    for j=1:n-k
                         if i==j+k
                              A1(i,j)=1;
                         end
                    end
               end
               B=[B,A1];
          end
     end
     end
     end
end
end
```

Figura 3-7 Relaciones del jugador 8 con el resto

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

C:\Documents and Settings\alumno\Mis documentos\MATLAB\bueno9.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function B=bueno9(n)
B=[];
for t=1:n-8
for h=1+t:n-7
for g=h+1:n-6
     for r=1+g:n-5
     for p=1+r:n-4
     for m=p+1:n-3
         for l=1+m:n-2
           for k=1+l:n-1
               A1=zeros(n,n-k);
               A1(l,:)=1;
               A1(k,:)=1;
               A1(m,:)=1;
               A1(p,:)=1;
               A1(r,:)=1;
               A1(g,:)=1;
               A1(h,:)=1;
               A1(t,:)=1;
               for i=k+1:n
                    for j=1:n-k
                         if i==j+k
                              A1(i,j)=1;
                         end
                    end
               end
               B=[B,A1];
          end
     end
     end
     end
end
end
end
end
```

Figura 3-8 Relaciones del jugador 9 con el resto

C:\Documents and Settings\alumno\Mis documentos\MATLAB\bueno10.m

File Edit Text Go Cell Tools Debug Desktop Window Help

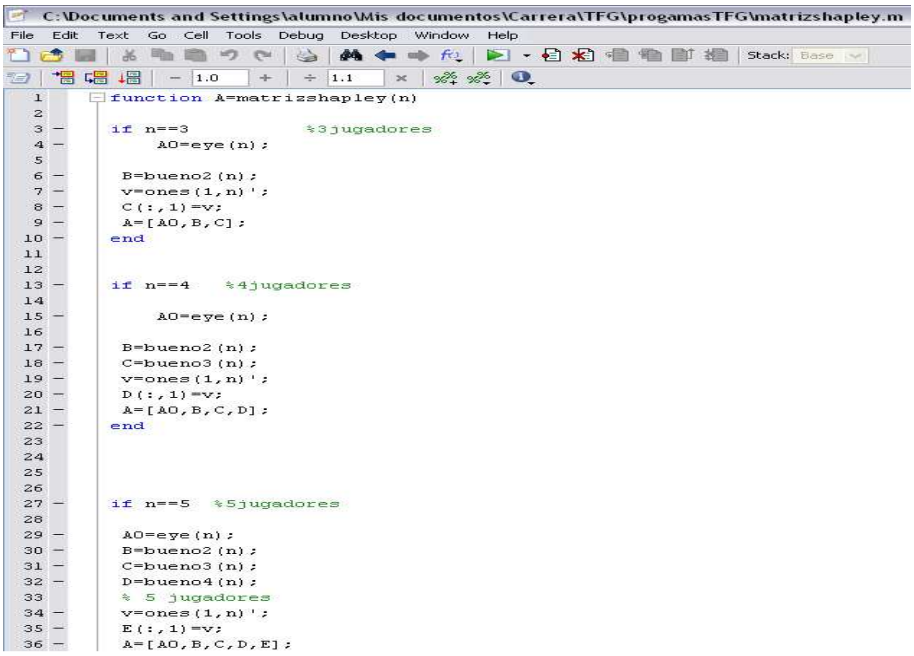
```
function B=bueno10(n)
B=[];
for d=1:n-9
for t=1+d:n-8
for h=1+t:n-7
for g=h+1:n-6
     for r=1+g:n-5
     for p=1+r:n-4
     for m=p+1:n-3
         for l=1+m:n-2
           for k=1+l:n-1
               A1=zeros(n,n-k);
               A1(l,:)=1;
               A1(k,:)=1;
               A1(m,:)=1;
               A1(p,:)=1;
               A1(r,:)=1;
               A1(g,:)=1;
               A1(h,:)=1;
               A1(t,:)=1;
               A1(d,:)=1;
               for i=k+1:n
                    for j=1:n-k
                         if i==j+k
                              A1(i,j)=1;
                         end
                    end
               end
               B=[B,A1];
          end
     end
     end
     end
end
end
end
end
```

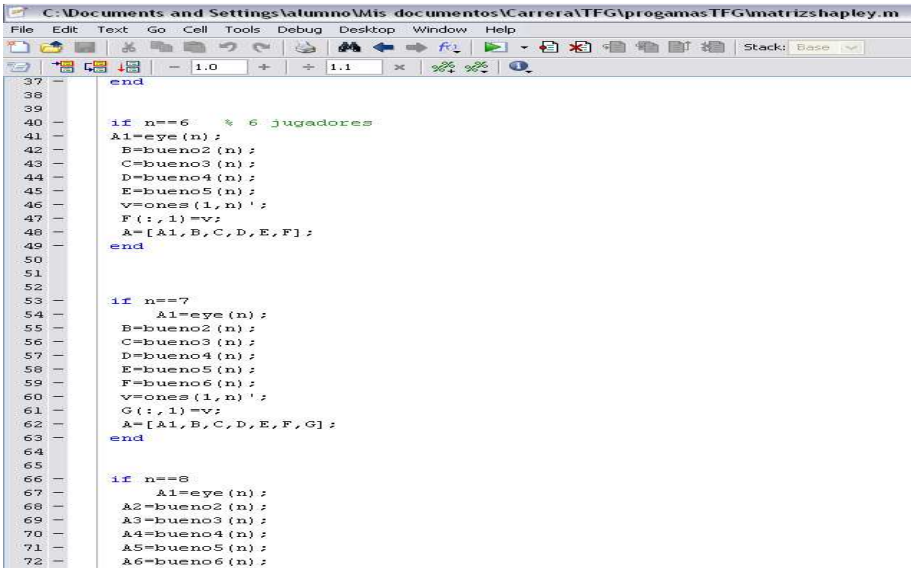
Figura 3-9 Relaciones del jugador 10 con el resto

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

-Función de cálculo de las matrices de pertenencia a soluciones

La siguiente función llama a las subfunciones definidas anteriormente y permite obtener la matriz de Shapley para “n” jugadores. La función pide como argumento de entrada el parámetro “n” que es el número de jugadores.





3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

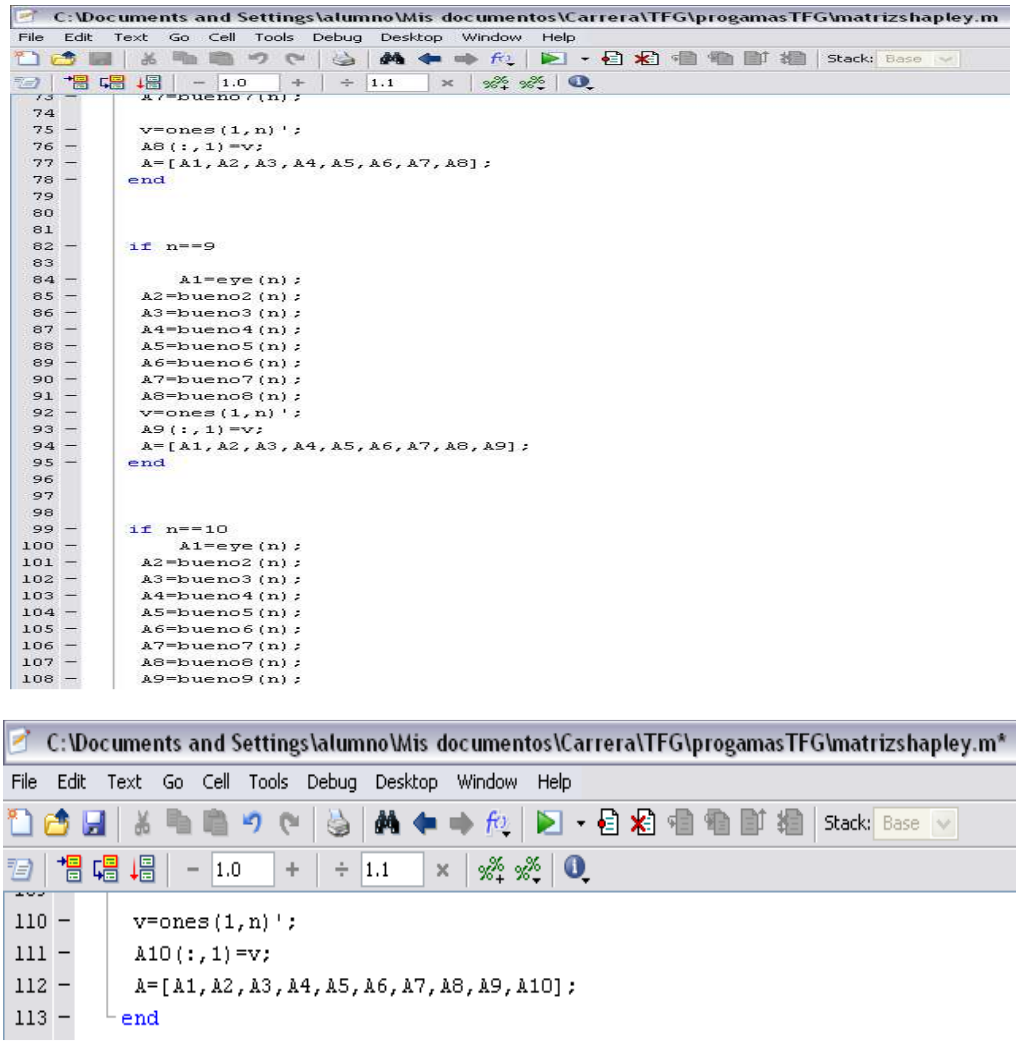


Figura 3-10 Programa que calcula la Matriz de Pertenencia a Coaliciones para n jugadores

Asi por ejemplo para 3 jugadores la Matriz de Pertenencia a Coaliciones quedaría de la siguiente forma:

	MATRIZ DE PERTENENCIA A COALICIONES PARA 3 JUGADORES							
	Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
Número de jugador	1	1	0	0	1	1	0	1
	2	0	1	0	1	0	1	1
	3	0	0	1	0	1	1	1

Figura 3-11 Matriz de Pertenencia a Coaliciones para 3 jugadores

Por tanto la Matriz de Pertenencia a Coaliciones es una matriz de unos y ceros la cual representa para cada columna los jugadores que intervienen en cada coalición. Esta matriz la necesitamos calcular para poder representar las ecuaciones de los modelos. Para estos modelos las restricciones se calculan multiplicando el vector "x" (solución del modelo) por cada columna de la matriz. De modo que obtendremos 2^{n-1}

restricciones para cada modelo. Para todas las Matrices de Pertenencia a Coaliciones el cálculo es similar. Primero se define una matriz identidad de tamaño el número de jugadores, a continuación se aplican los algoritmos bueno que van de 2 hasta N-1 jugadores, estos algoritmos devuelven para cada jugador el conjunto de coaliciones que tienen con los demás. Finalmente se añade una columna de unos para completar la última columna de la matriz.

3.1.2 Obtención de la función característica

Para este apartado usaremos el ejemplo de la bancarrota.
En este juego n acreedores deben repartirse el patrimonio neto de una empresa que se ha declarado en bancarrota. Se define con la siguiente función característica:

$$v(S) = \max\left\{0, E - \sum_{j \notin S} d_j\right\} \quad \forall S \subseteq N$$

Donde E es el patrimonio a repartir entre los jugadores y "d" es el vector de deudas que tenía la empresa con cada acreedor.
Para este trabajo hemos establecido el patrimonio a repartir entre los jugadores "E" como 100. En adelante "E=100".
En primer lugar se define el vector de deudas d:

-Primero se crea una tabla de números aleatorios entre 0 y 1 , de dimensión el número de jugadores .
-A continuación se define di como $200 * u_i / \sum_{i=1}^{n} ui$. Siendo ui el número aleatorio obtenido.
-Ahora para calcular el sumatorio de las j que no intervienen definimos la Matriz de Pertenencia a Coaliciones complementaria y definimos la función MAX (0, E – SUMAPRODUCTO (vector d, columna de la matriz complementaria)). Recorremos y obtenemos asi la función característica.

Cabe resaltar que el vector de deuda lo hemos creado de forma que la suma de todas sus componentes es 200. De este modo para 3 hasta 10 jugadores el vector de deudas di tendrá entre 3 y 10 componentes y el sumatorio de las componentes del vector de deudas para cada jugador será 200. Es decir para n=3…10 , $\sum_{i=1}^{n} di = 200$.
Para 3 hasta 10 jugadores se ha seguido este procedimiento. Por ejemplo para 3 jugadores este es el procedimiento empleado.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

Patrimonio	100

u1	0,2722
u2	0,0858
u3	0,7371
Suma ui	1,0951

d	49,717
	15,662
	134,62
Suma (di)	200

Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
A	1	0	0	1	1	0	1
	0	1	0	1	0	1	1
	0	0	1	0	1	1	1
V	0	0	4,789	0	31,16	73,63	100

Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
Acomp	0	1	1	0	0	1	0
	1	0	1	0	1	0	0
	1	1	0	1	0	0	0
V	0	0	4,789	0	31,16	73,63	100

Figura 3-12 Obtención de la función característica para 3 jugadores

3.2 SOLUCIONES DE JUEGOS COOPERATIVOS

Una vez que tenemos las matrices de Pertenencia a Coaliciones y los vectores de la función característica podemos pasar a resolver los modelos.

3.2.1 Modelo Least Core

Ahora se muestra su representación en LINGO.
-Juego de beneficio

Figura 3-13 Representación en Lingo del modelo Least Core para juegos de beneficio

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

- Juego de Coste

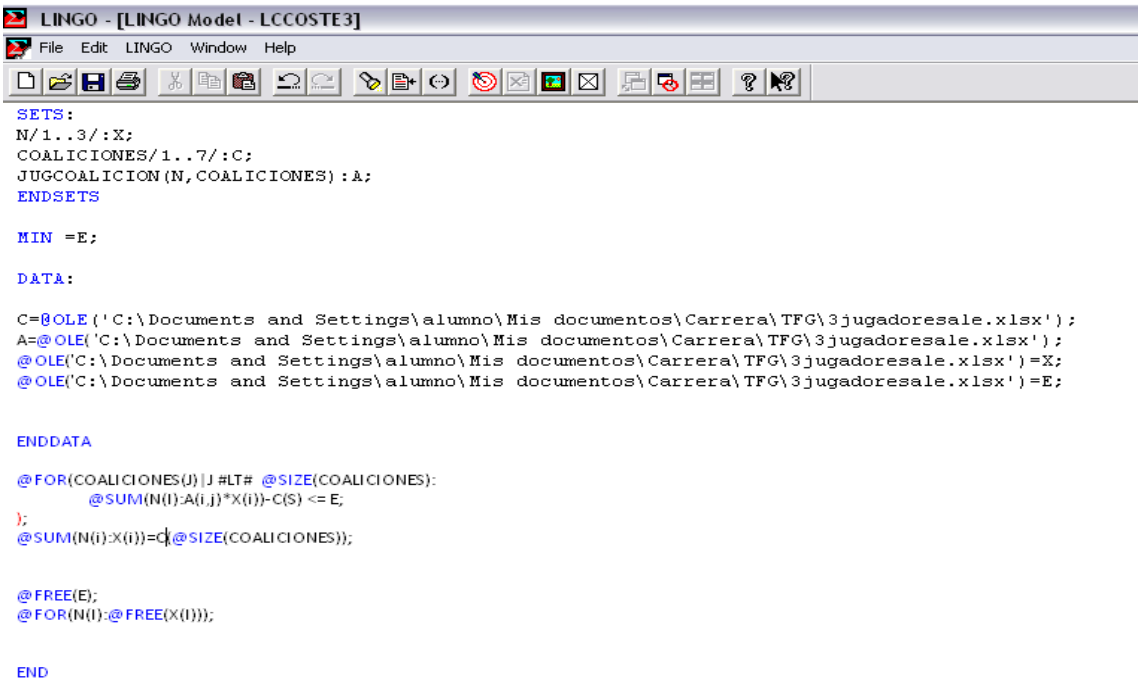


Figura 3-14 Representación en Lingo del modelo Least Core para juegos de coste

Este modelo en primer lugar recoge los datos V (función característica) y A (Matriz de Pertenencia a Coaliciones) guardadados en un fichero excel a través de la función @OLE. A continuación guarda en la misma hoja las soluciones del modelo X (vector de "n" componentes solución del problema) y E que es el exceso máximo.

La orden @SUM se usa para imponer que la suma de todas las componentes del vector solución debe ser igual al valor de la ultima componente del vector de la función característica.

Es necesario usar la orden FREE y aplicarla tanto para E como para X pues pueden tener resultados negativos.

Estos son los resultados que se vuelcan en el archivo Excel para el juego de beneficio para 3 jugadores.

Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
A	1	0	0	1	1	0	1
	0	1	0	1	0	1	1
	0	0	1	0	1	1	1
V	0	0	4,7893057	0	31,161178	73,628128	100

X	13,185936	55,652887	31,161178
Emin	-13,185936		

Figura 3-15 Resultados modelo Least Core 3 jugadores

En este gráfico se adjuntan los resultados obtenidos con el modelo Least-Core para 3 hasta 10 jugadores.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

		RESULTADOS LEAST-CORE										
	REPARTO	1	2	3	4	5	6	7	8	9	10	Emax
NÚMERO DE JUGADORES	3	13,2	55,7	31,2								-13,2
	4	39,6	35,9	12,3	12,3							-12,3
	5	6,55	37,7	40,2	8,99	6,55						-6,55
	6	23,5	47,1	15,1	4,75	4,75	4,75					-4,75
	7	8,33	6,66	60,2	6,19	6,19	6,19	6,19				-6,19
	8	2,78	30,9	44,6	2,13	5,52	2,13	2,13	9,84			-2,13
	9	11,9	16,2	33,5	32,9	0,46	0,46	3,64	0,46	0,46		-0,46
	10	3,3	37,3	2,25	2,25	2,25	29,6	2,25	2,25	16,3	2,25	-2,25

Figura 3-16 Resultados modelo Least Core para 3 hasta 10 jugadores

3.2.2 Modelo de Drechsel

Esta es la representación del Modelo de Drechsel en Lingo utilizado para tres jugadores:

-Juego de beneficio

LINGO - [LINGO Model - Drechsel3]

File Edit LINGO Window Help

```
SETS:
N/1..3/:X;
COALICIONES/1..7/:V;
JUGCOALICION(N,COALICIONES):A;
ENDSETS

MAX =UN;

DATA:

V=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\Drechsel3.xlsx');
A=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\Drechsel3.xlsx');
@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\Drechsel3.xlsx')=X;
@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\Drechsel3.xlsx')=UN;


ENDDATA

@FOR(COALICIONES(J)|J #LT# @SIZE(COALICIONES):
        @SUM(N(I):A(i,j)*X(i)) >= UN*V(J);
);
@SUM(N(i):X(i))=V(@SIZE(COALICIONES));


@FREE(UN);
@FOR(N(I):@FREE(X(I)));


END
```

Figura 3-17 Representación en Lingo del modelo Drechsel para juegos de beneficio

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

-Juego de coste

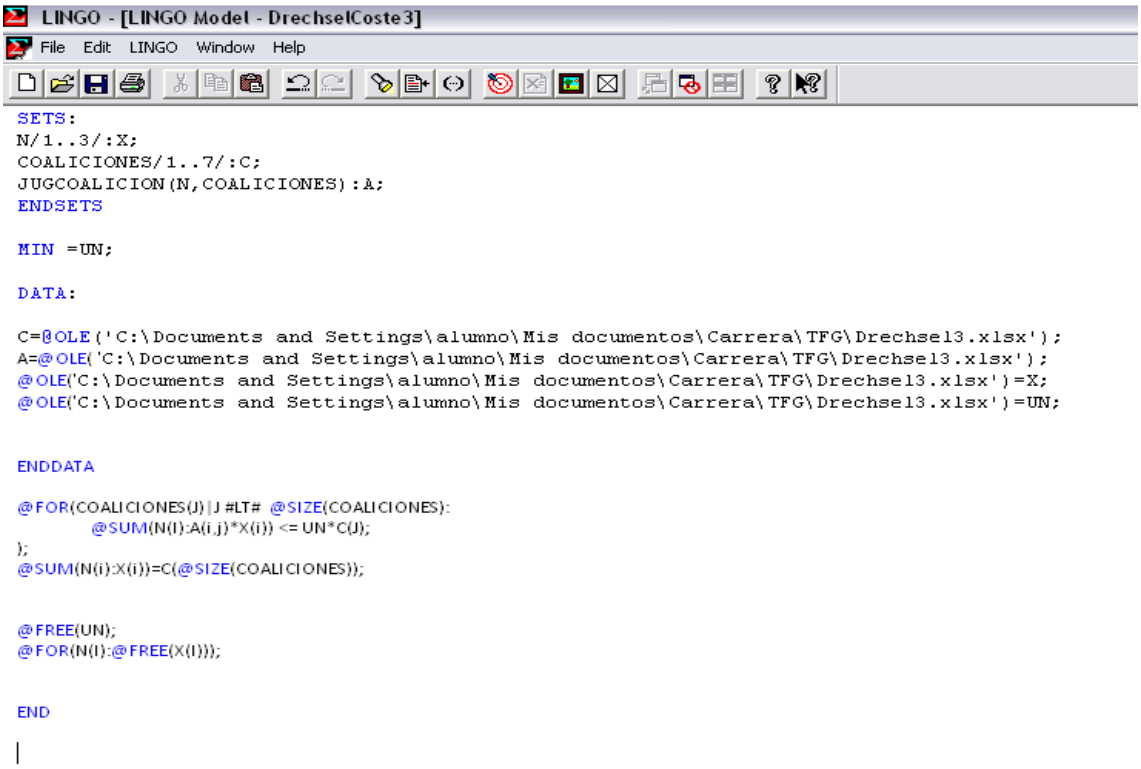


Figura 3-18 Representación en Lingo del modelo Drechsel para juegos de coste

Este modelo es similar al anterior con la diferencia que ahora queremos maximizar el valor de la variable "UN".

Estos son los resultados obtenidos para tres jugadores, para un juego de beneficio. Al igual que antes la solución se vuelca en el archivo Excel.

Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
A	1	0	0	1	1	0	1
	0	1	0	1	0	1	1
	0	0	1	0	1	1	1
V	0	0	4,7893057	0	31,161178	73,628128	100

X	0	57,677618	42,322382
UN	1,3581766		

Figura 3-19 Resultados modelo Drechsel para 3 jugadores

Estos son los resultados para 3 hasta 10 jugadores.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

		RESULTADOS DRECHSEL										
	REPARTO	1	2	3	4	5	6	7	8	9	10	UN
NÚMERO DE JUGADORES	3	0	57,7	42,3								1,36
	4	36,2	38,4	25,3	0							1,32
	5	0	0	38,8	30,6	30,6						1,15
	6	0	38,9	39,1	0	0	22,1					1,05
	7	0	0	61,7	24,1	0	9,49	4,75				1,14
	8	0,68	30,1	0	0	42,9	12,8	13,6	0			1,04
	9	11,5	0	2,79	36,8	11	37,9	0	0	0		1,01
	10	6,4	33,9	0	0	0	28,6	3,95	1,21	14,7	11,2	1,05

Figura 3-20 Resultados modelo Drechsel para 3 hasta 10 jugadores

3.2.3 Modelo Lexicográfico

Representación en Lingo

-Juego de beneficio

LINGO - [LINGO Model - lexico3_2]

File Edit LINGO Window Help

```
SETS:
N/1..3/:X;
COALICIONES/1..7/:V;
JUGCOALICION(N,COALICIONES):A;
ENDSETS

MAX =Emin;

DATA:

V=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lexico3_2.xlsx');
A=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lexico3_2.xlsx');
E=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\3jugadoresale.xlsx');

@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lingo-excel\lexico3_2.xlsx')=X;
@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lingo-excel\lexico3_2.xlsx')=Emin;


ENDDATA

@FOR(COALICIONES(J)|J #LT# @SIZE(COALICIONES):
        V(j)-@SUM(N(I):A(i,j)*X(i)) <= E;
);


@FOR(COALICIONES(J)|J #LT# @SIZE(COALICIONES):
        V(j)-@SUM(N(I):A(i,j)*X(i)) >=Emin;
);


@SUM(N(i):X(i))=V(@SIZE(COALICIONES));

@FREE(Emin);
@FOR(N(I):@FREE(X(I)));



END
```

Figura 3-21 Representación en Lingo del modelo Lexicográfico para juegos de beneficio

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

-Juego de coste

LINGO - [LINGO Model - LexicoCoste3]

File Edit LINGO Window Help

```
SETS:
N/1..3/:X;
COALICIONES/1..7/:C;
JUGCOALICION(N,COALICIONES):A;
ENDSETS

MAX =Emin;

DATA:

C=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lexico3_2.xlsx');
A=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lexico3_2.xlsx');
E=@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\3jugadoresale.xlsx');

@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lingo-excel\lexico3_2.xlsx')=X;
@OLE('C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\lingo-excel\lexico3_2.xlsx')=Emin;


ENDDATA

@FOR(COALICIONES(J)|J #LT# @SIZE(COALICIONES):
        @SUM(N(I):A(i,j)*X(i))-C(j) <= E;
);


@FOR(COALICIONES(J)|J #LT# @SIZE(COALICIONES):
        @SUM(N(I):A(i,j)*X(i)) -C(j)>=Emin;
);


@SUM(N(i):X(i))=V(@SIZE(COALICIONES));

@FREE(Emin);
@FOR(N(I):@FREE(X(I)));



END
```

Figura 3-22 Representación en Lingo del modelo Lexicográfico para juegos de coste

Se observa que ε^{max} (que en el modelo viene expresado como E) se obtiene del fichero Excel correspondiente al modelo Least Core. Los datos se vuelcan en el archivo Excel correspondiente al modelo.

Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
A	1	0	0	1	1	0	1
	0	1	0	1	0	1	1
	0	0	1	0	1	1	1
V	0	0	4,7893057	0	31,161178	73,628128	100

X	13,185936	34,419411	52,394653
Emin	-47,605347		

Figura 3-23 Resultados modelo Lexicográfico para 3 jugadores

En la siguiente tabla se adjuntan los resultados obtenidos con el modelo lexicográfico, para el juego de beneficio.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

		RESULTADOS LEXICOGRÁFICO										
	REPARTO	1	2	3	4	5	6	7	8	9	10	Emin
NÚMERO DE JUGADORES	3	13,2	34,4	52,4								-48
	4	35,5	40	12,3	12,3							-48
	5	6,55	21,8	21,8	28,4	21,4						-50
	6	18	26	22	18	4,75	11,3					-49
	7	12,5	6,66	34,3	15,7	6,19	8,99	15,7				-50
	8	2,69	14	23,2	2,13	22	15,4	11,8	8,75			-50
	9	5,05	6	19,4	19,9	4,53	20,5	0,69	0,46	23,4		-50
	10	5,2	19,7	2,25	18,3	14,8	16,1	4,26	2,94	9,05	7,41	-50

Figura 3-24 Resultados modelo Lexicográfico para 3 hasta 10 jugadores

3.2.4 Método del valor de Shapley

El procedimiento que se describe vale tanto para juegos de beneficio como para juegos de coste.

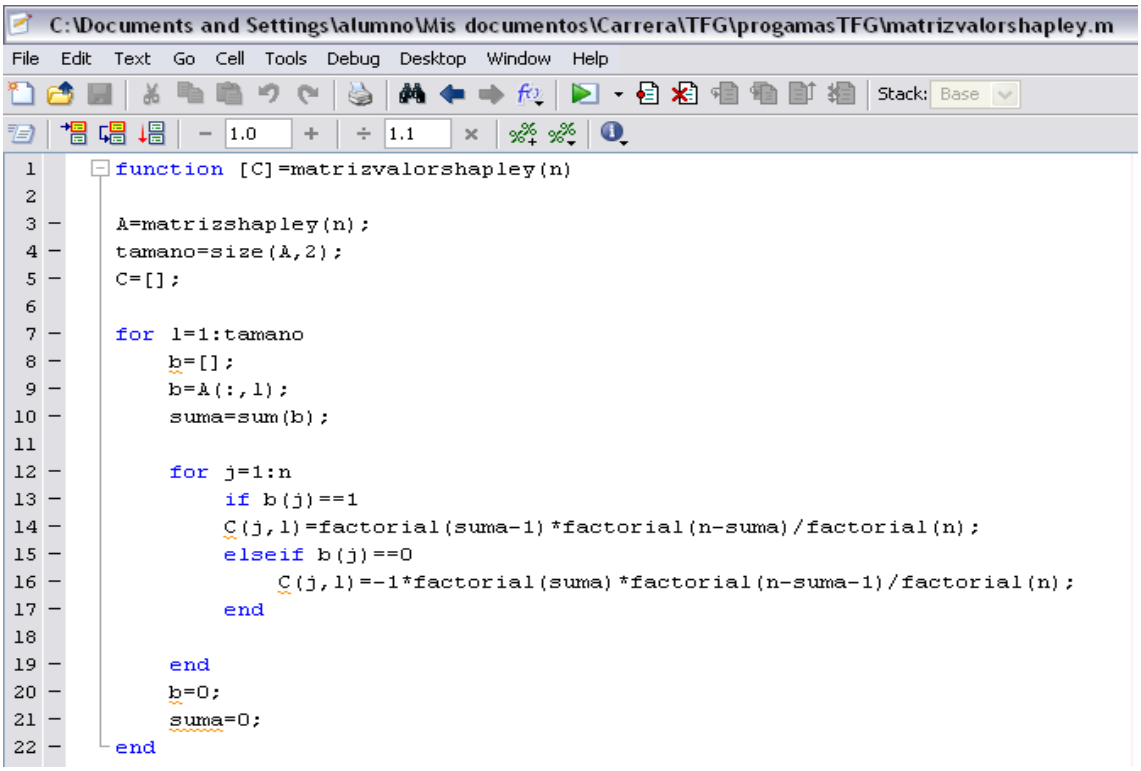
-Matriz de Shapley programada con Matlab

En este programa se recorre un bucle for para cada columna . Para cada iteración de este bucle se crea un vector v cuyas componentes son cada columna de la matriz. Almacena la variable “suma” que es la suma de las componentes del vector v. Ahora para cada componente del vector v pregunta si es un “1” o un “0” para asi definir el elemento de la nueva matriz de una forma u otra.

Si el elemento del vector “v” es “1” aplica la fórmula (suma-1)!(n-suma)!/n!; si el elemento es “0” aplica la fórmula –suma!(n-suma-1)!/n!.

Este es el programa matlab utilizado para calcular la matriz asociada al valor de Shapley recpecto de la base canónica.

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES




```
function [C]=matrizvalorshapley(n)

A=matrizshapley(n);
tamano=size(A,2);
C=[];

for l=1:tamano
    b=[];
    b=A(:,l);
    suma=sum(b);

    for j=1:n
        if b(j)==1
        C(j,l)=factorial(suma-1)*factorial(n-suma)/factorial(n);
        elseif b(j)==0
            C(j,l)=-1*factorial(suma)*factorial(n-suma-1)/factorial(n);
        end

    end
    b=0;
    suma=0;
end
```

Figura 3-25 Programa modelo Shapley para n jugadores

-Representación de la Matriz de Shapley para 3 jugadores.

Abajo se representa la matriz de Shapley donde arriba en la fila "coaliciones" se indican los jugadores que participan en cada coalición. Como se ha explicado anteriormente, dependiendo de si el elemento que se esta recorriendo es 1 o 0 se procede con un cálculo o con otro.

	MATRIZ DE SHAPLEY PARA 3 JUGADORES						
Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
A	0,3333333	-0,166667	-0,166667	0,1666667	0,1666667	-0,333333	0,3333333
	-0,166667	0,3333333	-0,166667	0,1666667	-0,333333	0,1666667	0,3333333
	-0,166667	-0,166667	0,3333333	-0,333333	0,1666667	0,1666667	0,3333333

Figura 3-26 Matriz de Shapley para 3 jugadores

-Resultados del Método de Shapley para 3 jugadores

3. RUTINAS DE CÁLCULO Y ANÁLISIS DE SOLUCIONES

Coaliciones	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
A	1	0	0	1	1	0	1
	0	1	0	1	0	1	1
	0	0	1	0	1	1	1
V	0	0	4,7893057	0	31,161178	73,628128	100

X	13,185936	34,419411	52,394653
Emin	-47,605347		

Figura 3-27 Resultados modelo Shapley para n jugadores

- Resultados para 3 hasta 10 jugadores.

		RESULTADOS SHAPLEY									
	REPARTO	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
NÚMERO DE JUGADORES	3	13,19	34,42	52,39							
	4	25,94	39,98	21,82	12,26						
	5	6,546	22,11	23,39	28,09	19,85					
	6	14,13	25,93	22,43	18,03	4,746	14,73				
	7	12,11	6,428	33,21	16,74	6,194	10,35	14,96			
	8	2,455	16,52	23,34	2,13	22,68	18,26	8,622	5,985		
	9	6,165	8,344	16,99	18,67	5,901	19,26	2,051	0,461	22,16	
	10	5,197	19,69	2,252	18,34	14,77	16,09	4,258	2,945	9,045	7,413

Figura 3-28 Resultados modelo Shapley para 3 hasta 10 jugadores

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

4.1 VALORES NÚMERICOS Y REPRESENTACIÓN GRÁFICA PARA 3 HASTA 10 JUGADORES

Las siguientes gráficas representan los resultados obtenidos al aplicar los modelos Least Core, Lexicográfico, Drechsel y Shapley para las coaliciones calculadas anteriormente para 3 hasta 10 jugadores.

-3 Jugadores

	3 JUGADORES		
	1	2	3
LEAST_CORE	13,186	55,653	31,161
DRECHSEL	0	57,678	42,322
LEXICOGRÁFICO	13,186	34,419	52,395
SHAPLEY	13,186	34,419	52,395

Figura 4-1 Comparativa de resultados para 3 jugadores

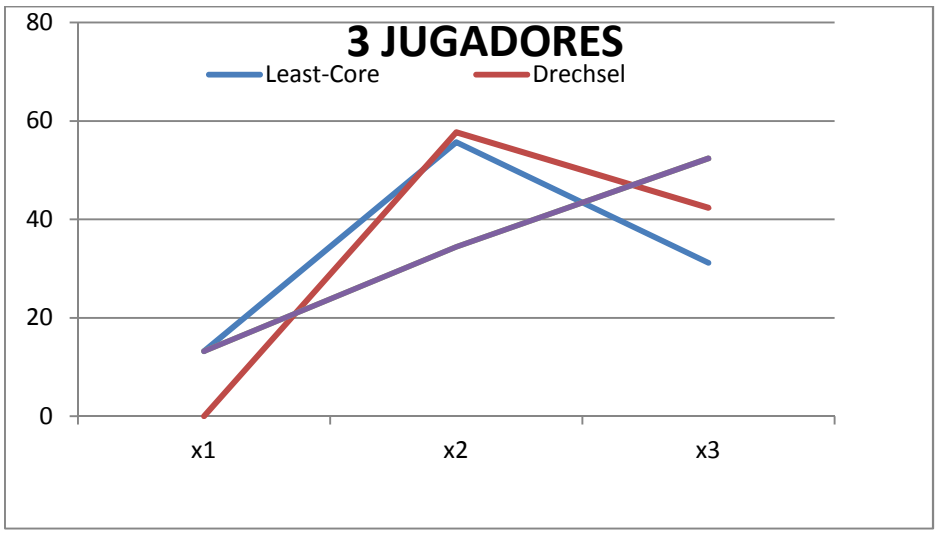


Figura 4-2 Gráfica Comparativa de resultados para 3 jugadores

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

Para tres jugadores los métodos Shapley y Lexicográfico dan soluciones idénticas mientras que los modelos Least Core y Drechsel aportan soluciones parecidas entre sí.

Se observa una tendencia al alza en las 4 gráficas.

-4 Jugadores

-

	4 JUGADORES			
	1	2	3	4
LEAST_CORE	39,612	35,866	12,261	12,261
DRECHSEL	36,237	38,423	25,34	0
LEXICOGRÁFICO	35,5	39,978	12,261	12,261
SHAPLEY	25,936	39,978	21,824	12,261

Figura 4-3 Comparativa de resultados para 4 jugadores

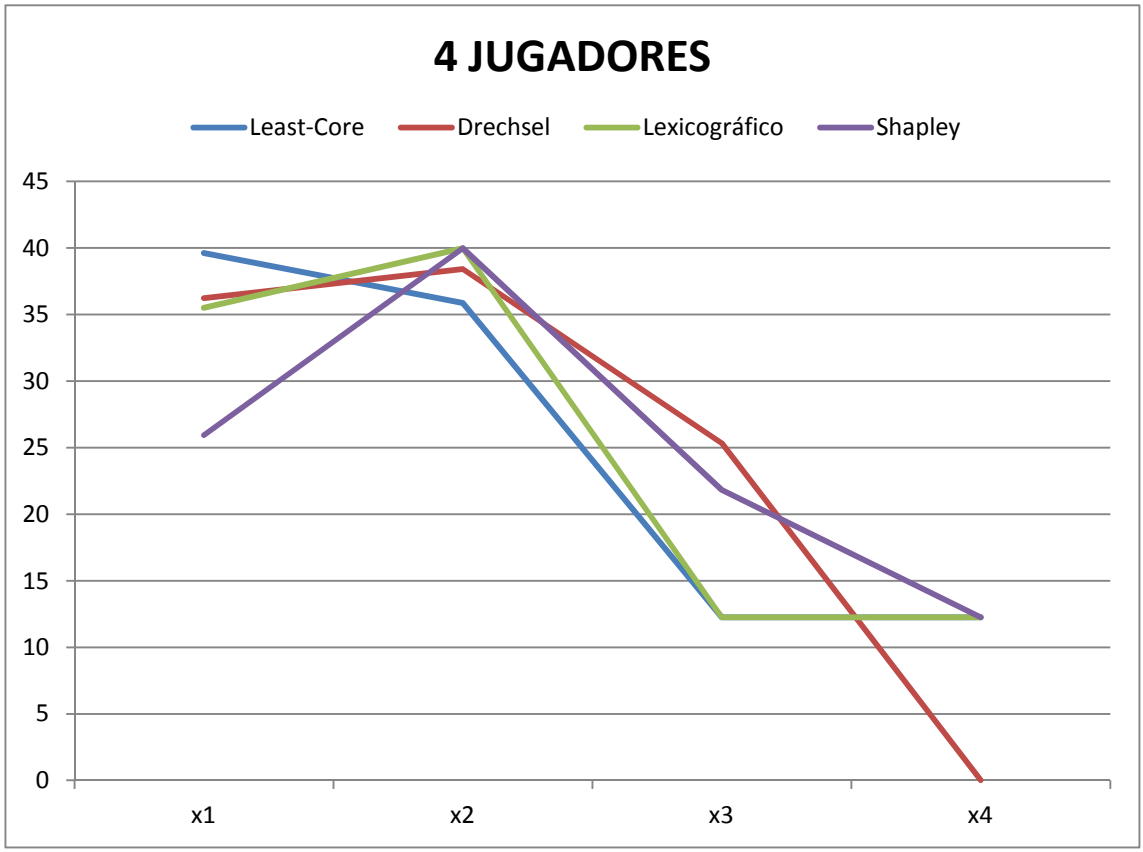


Figura 4-4 Gráfica Comparativa de resultados para 4 jugadores

Para 4 jugadores se observa una tendencia a la baja en las 4 gráficas. Igualmente se aprecia que las 4 gráficas son muy parecidas entre sí.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

-5 Jugadores

	5 JUGADORES				
	1	2	3	4	5
LEAST_CORE	6,546	37,682	40,237	8,9899	6,546
DRECHSEL	0	0	38,766	30,608	30,626
LEXICOGRÁFICO	6,546	21,831	21,831	28,377	21,415
SHAPLEY	6,546	22,114	23,391	28,095	19,854

Figura 4-5 Comparativa de resultados para 5 jugadores

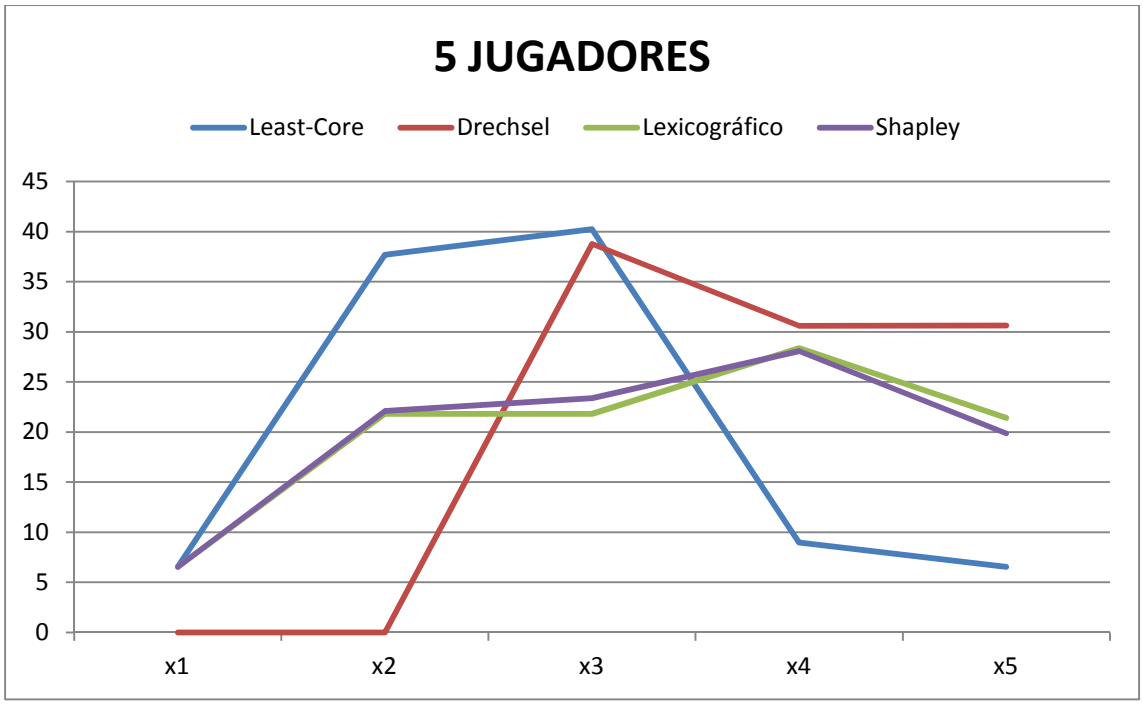


Figura 4-6 Gráfica Comparativa de resultados para 5 jugadores

A diferencia de los casos anteriores, para 5 jugadores las gráficas de Least-Core y Drechsel son diferentes a las otras. Estas últimas ofrecen soluciones parecidas y equilibradas (el reparto es en cierta medida equitativo) mientras que las gráficas de Least-Core y Drechsel dan repartos muy desiguales.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

-6 Jugadores

	6 JUGADORES					
	1	2	3	4	5	6
LEAST_CORE	23,522	47,117	15,123	4,7458	4,7458	4,7458
DRECHSEL	0	38,868	39,074	0	0	22,058
LEXICOGRÁFICO	17,951	26,013	21,997	17,951	4,7458	11,342
SHAPLEY	14,134	25,931	22,429	18,032	4,7458	14,728

Figura 4-7 Comparativa de resultados para 6 jugadores

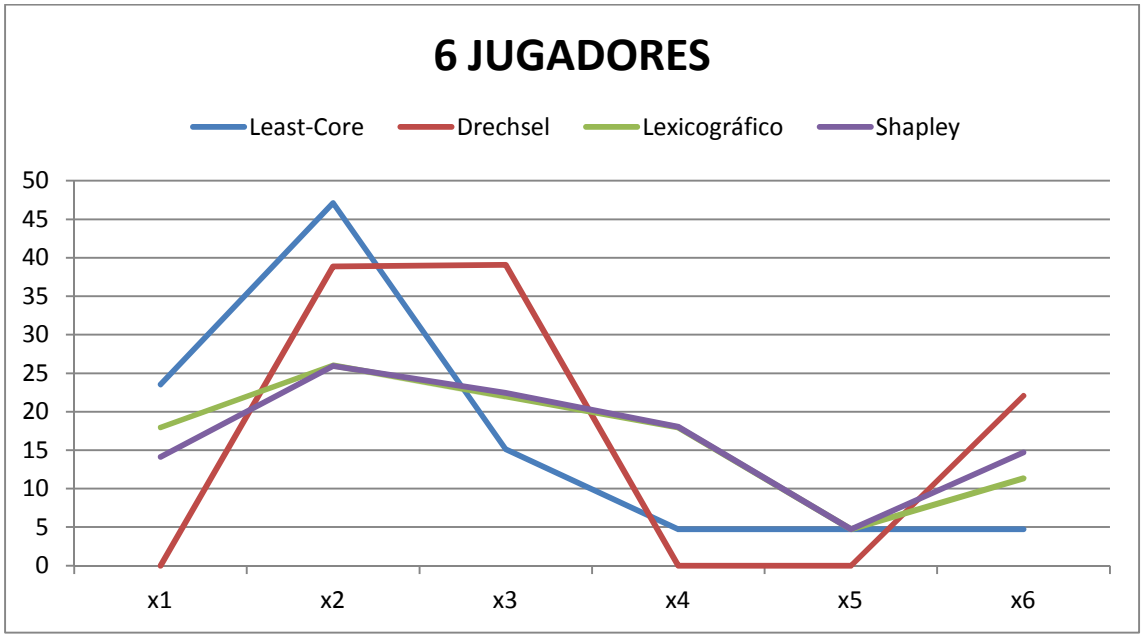


Figura 4-8 Gráfica Comparativa de resultados para 6 jugadores

Esta gráfica ofrece resultados parecidos a la anterior. Los modelos Shapley y Lexicográfico ofrecen soluciones muy parecidas pero su tendencia es a la baja, a diferencia del caso anterior. Las gráficas de Least-Core ofrecen repartos muy desiguales pero son más parecidas entre sí que en la anterior gráfica.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

-7 Jugadores

	7 JUGADORES						
	1	2	3	4	5	6	7
LEAST_CORE	8,3283	6,6628	60,234	6,1937	6,1937	6,1937	6,1937
DRECHSEL	0	0	61,681	24,081	0	9,4899	4,7484
LEXICOGRÁFICO	12,499	6,6628	34,336	15,657	6,1937	8,9943	15,657
SHAPLEY	12,11	6,4283	33,214	16,743	6,1937	10,351	14,96

Figura 4-9 Comparativa de resultados para 7 jugadores

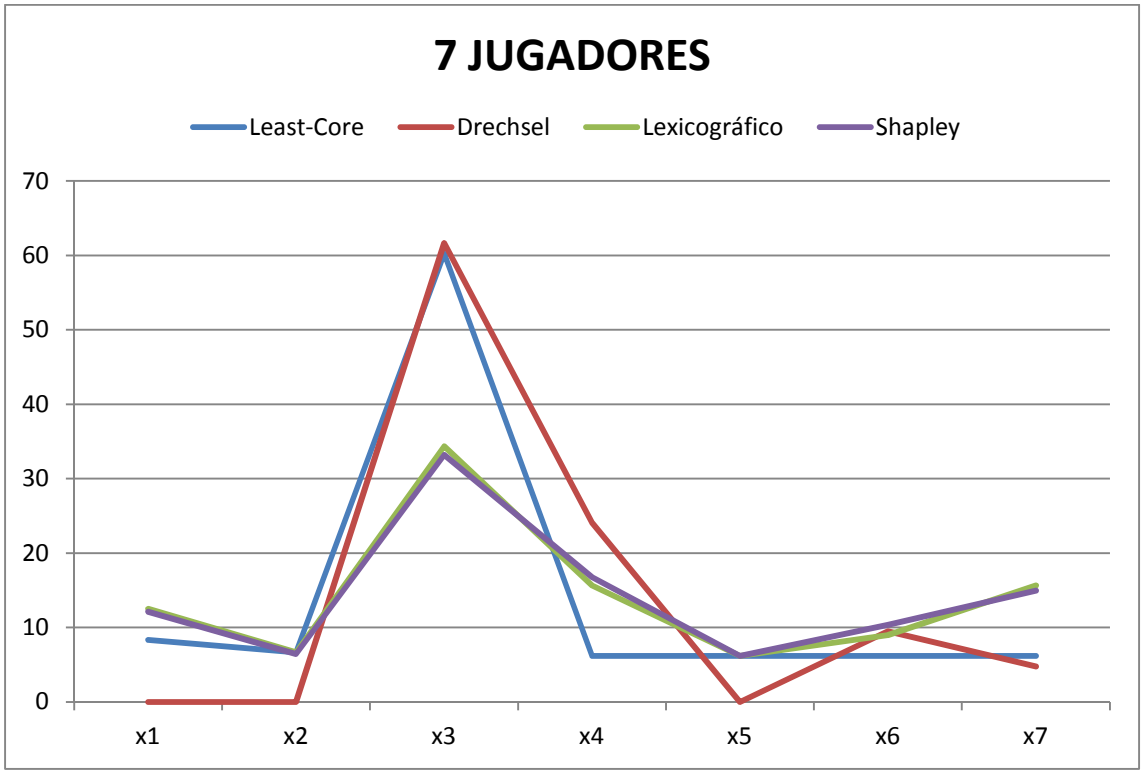


Figura 4-10 Gráfica Comparativa de resultados para 7 jugadores

En este caso, los modelos Shapley y Lexicográfico ofrecen resultados más desigules que en gráficas anteriores. El jugador 3 sale bastante más beneficiado que el resto. Las gráficas del modelo Least-Core y Drechsel ofrecen resultados muy parecidos entre sí, más que en casos anteriores.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

-8 Jugadores

	8 JUGADORES							
	1	2	3	4	5	6	7	8
LEAST_CORE	2,7803	30,915	44,556	2,1302	5,5174	2,1302	2,1302	9,8406
DRECHSEL	0,679	30,065	0	0	42,929	12,764	13,562	0
LEXICOGRÁFICO	2,6912	14,027	23,23	2,1302	21,993	15,373	11,805	8,7503
SHAPLEY	2,4552	16,522	23,343	2,1302	22,68	18,261	8,6225	5,9854

Figura 4-11 Comparativa de resultados para 8 jugadores

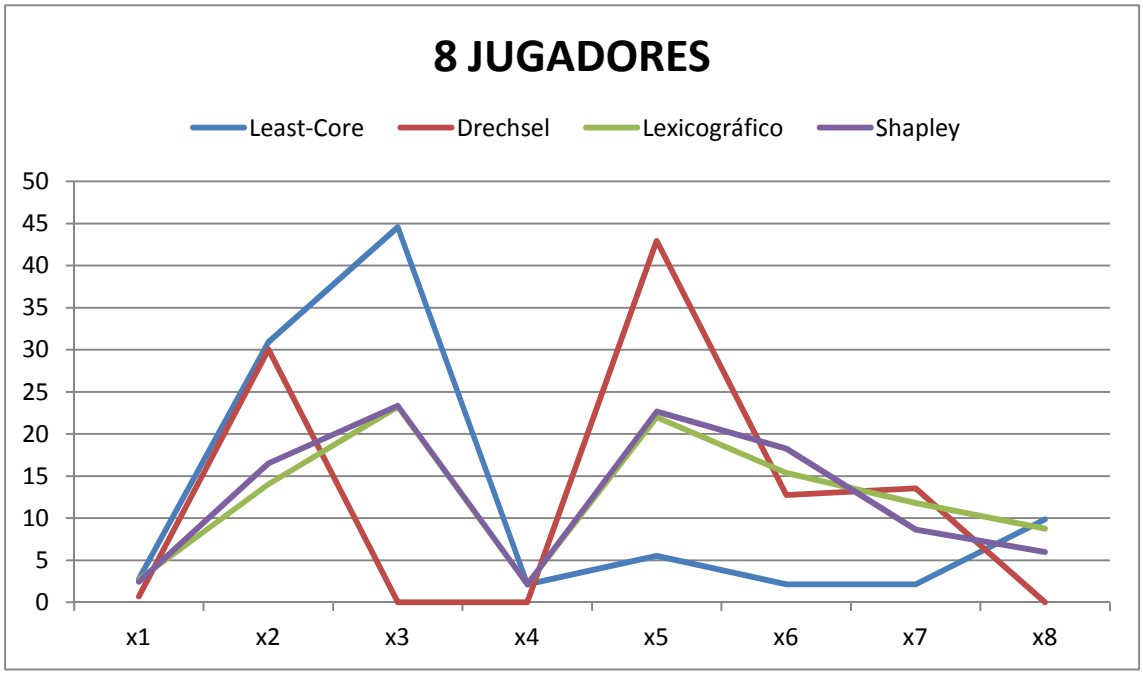


Figura 4-12 Gráfica Comparativa de resultados para 8 jugadores

Se puede apreciar que las gráficas del modelo Shapley y Lexicográfico ofrecen gráficas prácticamente simétricas que evidencian un reparto desigual, comparado con casos anteriores. Por primera vez, las gráficas de los modelos Least-Core y Drechsel son bastante diferentes entre sí.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

-9 Jugadores

	9 JUGADORES								
	1	2	3	4	5	6	7	8	9
LEAST_CORE	11,869	16,227	33,513	32,906	0,4607	0,4607	3,6419	0,4607	0,4607
DRECHSEL	11,515	0	2,7917	36,766	10,983	37,945	0	0	0
LEXICOGRÁFICO	5,0504	5,9992	19,414	19,875	4,5251	20,548	0,694	0,4607	23,433
SHAPLEY	6,1651	8,3438	16,987	18,674	5,9014	19,258	2,0513	0,4607	22,158

Figura 4-13 Comparativa de resultados para 9 jugadores

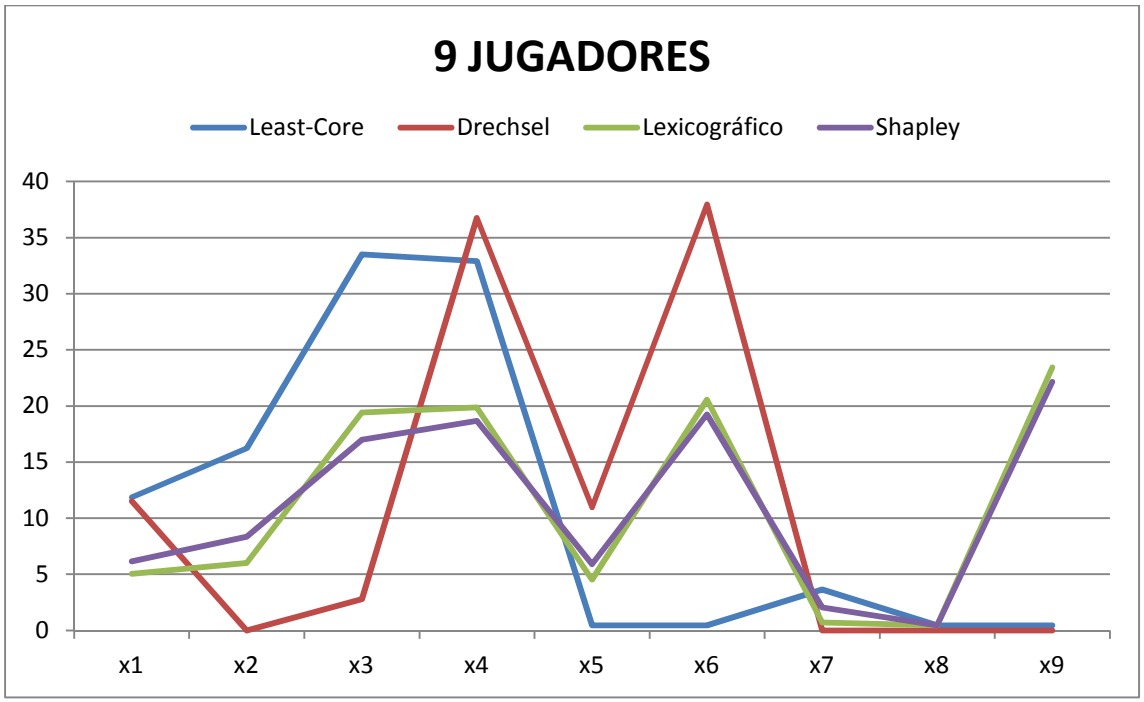


Figura 4-14 Gráfica Comparativa de resultados para 9 jugadores

Para este caso los modelos Shapley y Lexicográfico ofrecen soluciones parecidas al caso anterior. El reparto es más desigual pues el jugador 8 apenas recibe nada. Las gráficas de los otros modelos, al igual que en el caso anterior, son muy diferentes.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

-10 Jugadores

	10 JUGADORES									
	1	2	3	4	5	6	7	8	9	10
LEAST_CORE	3,3007	37,307	2,2519	2,2519	2,2519	29,595	2,2519	2,2519	16,286	2,2519
DRECHSEL	6,3953	33,908	0	0	0	28,632	3,9464	1,2052	14,696	11,217
LEXICOGRÁFICO	3,7531	39,559	4,5038	2,2519	2,2519	2,2519	8,2725	5,6548	18,538	15,215
SHAPLEY	5,1969	19,69	2,2519	18,339	14,774	16,087	4,2577	2,945	9,0452	7,4129

Figura 4-15 Comparativa de resultados para 10 jugadores

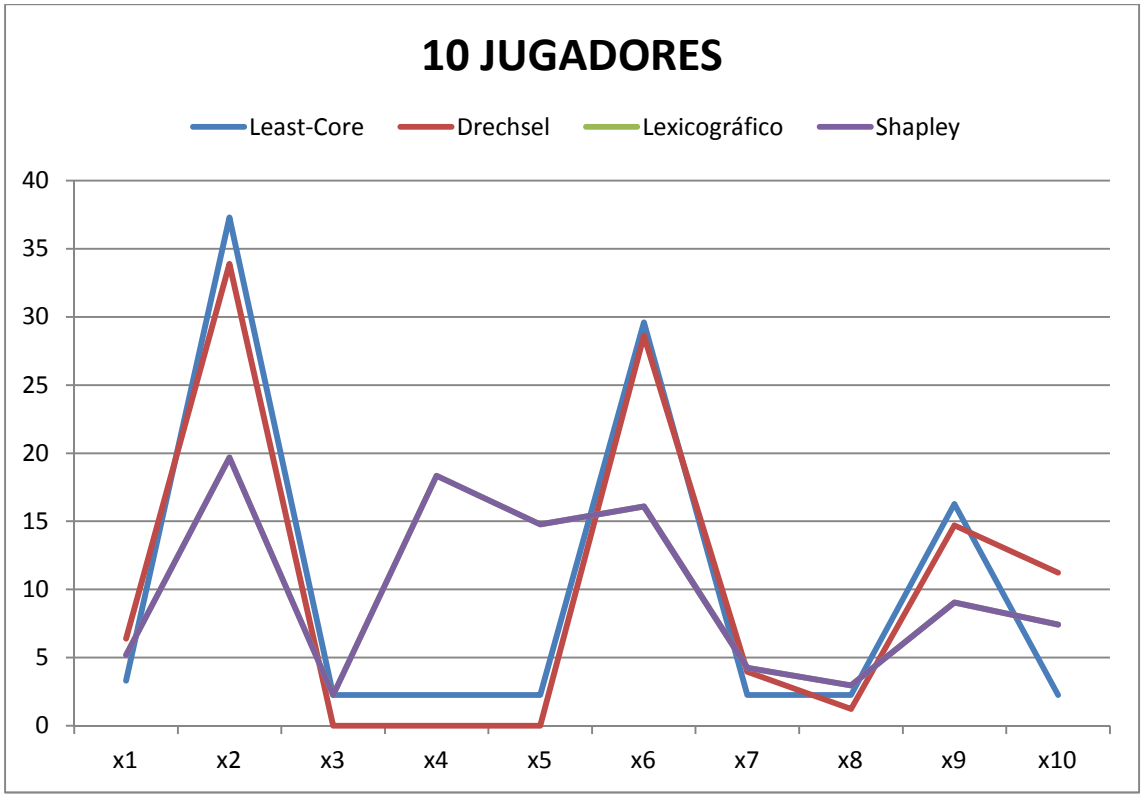


Figura 4-16 Gráfica Comparativa de resultados para 10 jugadores

Este es el único caso (aparte de para 3 jugadores) en el que los modelos Lexicográfico y Shapley ofrecen soluciones idénticas. Cabe añadir que el reparto para estas dos gráficas es bastante más equitativo que en casos anteriores. Para 8 y para 9 jugadores las otras dos gráficas eran bastante diferentes. Curiosamente, en este caso estas dos gráficas ofrecen resultados muy parecidos, aunque el reparto sigue siendo bastante desigual.

4.2 FUNCIONES DE COMPROBACIÓN DE SOLUCIONES

4.2.1 Pertenencia al core

-Juegos de beneficio

Para un juego de beneficio una solución pertenece al core si la suma de sus componentes es igual a la última componente de su vector característico y todos los sumatorios de las soluciones ”x” multiplicadas por cada columna de la Matriz de Pertenencia a Coaliciones es mayor que el valor de la componente del vector de la función característica correspondiente a dicha columna.

C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\progamasTFG\corebuenobeneficio.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function [S,b]=corebuenobeneficio(v,x)
tamano1=size(v,2);
tamano2=size(x,2);

     suma=0;
     for i=1:tamano2
          suma=suma+x(i);
     end
     A=matrizshapley(tamano2);
     flag=1;
     k=1;
     b=[];

     for i=1 :tamano1
        if (A(:,i))'*x'<v(i)
             flag=0;
             b(k)=v(i)-(A(:,i))'*x';
        end
        k=k+1;
     end

     if suma==v(tamano1)
          P=1;
     else
          P=0;
     end

     if P==1 & flag==1
          S=1;
     else
          S=0;
     end
```

Figura 4-17 Programa pertencia al Core (Juegos beneficio)

En este algoritmo si alguna vez no se cumple la condición de pertenencia al core a la variable bandera se le asigna el valor 0 y el algoritmo devuelve que la solución no pertenece al core (S=0).

-Juego de coste

Para un juego de coste una solución pertenece al core si la suma de sus componentes es igual a la última componente de su vector característico y todos los sumatorios de las soluciones “x” multiplicadas por cada columna de la Matriz de Pertenencia a Coaliciones son menores o iguales que el valor de la componente del vector de la función característica correspondiente a dicha columna.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\progamasTFG\corebuenocoste.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

Stack: Base

```
function [S,b]=corebuenocoste(v,x)
tamano1=size(v,2);
tamano2=size(x,2);
     suma=0;
     for i=1:tamano2
          suma=suma+x(i);
     end
     A=matrizshapley(tamano2);
     flag=1;
     k=1;
     b=[];
     for i=1 :tamano1
         if (A(:,i))'*x'>v(i)
              flag=0;
              b(k)=(A(:,i))'*x'-v(i);
         end
         k=k+1;
     end

     if suma==v(tamano1)
          P=1;
     else
          P=0;
     end

     if P==1 & flag==1
          S=1;
     else
          S=0;
     end
```

Figura 4-18 Programa pertencia al Core (Juegos coste)

En este algoritmo si alguna vez no se cumple la condición de pertenencia al core a la variable bandera se le asigna el valor 0 y el algoritmo devuelve que la solución no pertenece al core (S=0).

4.2.2 Representación gráfica del core y de las soluciones de los modelos para 3 y 4 jugadores

Para 3 y 4 jugadores el Core puede representarse gráficamente. Se muestra a continuación el código de Matlab empleado para obrener dicha representación para 3 jugadores. En primer lugar se declara el vector de la función característica para 3 jugadores. A continuación se crean los vectores “punto1”, “punto2”,”punto3” y “punto4” que almacenan el resultado del reparto obtenido con los 4 modelos estudiados. Ahora se representa el Core con la función “Coreset”. Para representar las soluciones obtenidas con los modelos usamos el comando “imputation3plot”.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

```
b3 =

         0         0    4.7893         0   31.1612   73.6281  100.0000

>> punto1

punto1 =

   13.1859   55.6529   31.1612

>> punto2

punto2 =

         0   57.6776   42.3224

>> punto3

punto3 =

   13.1859   34.4194   52.3947

>> punto4

punto4 =

   13.1859   34.4194   52.3947

>> imputationset(b3),hold on,axis(axis)
>> coreset(b3)
>> imputation3plot(b3,punto1,'*b')
>> imputation3plot(b3,punto2,'*c')
>> imputation3plot(b3,punto3,'*g')
>> imputation3plot(b3,punto4,'*m')
```

Figura 4-19 Programa para representar el Core para 3 jugadores

-3 JUGADORES

La zona sombreada de la siguiente figura representa el Core para 3 jugadores. Puede comprobarse que las soluciones ofrecidas por los 4 modelos están dentro del Core. Como la solución para el modelo Lexicográfico y el modelo Shapley es la misma, el mismo punto verde corresponde a la solución para ambos modelos.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

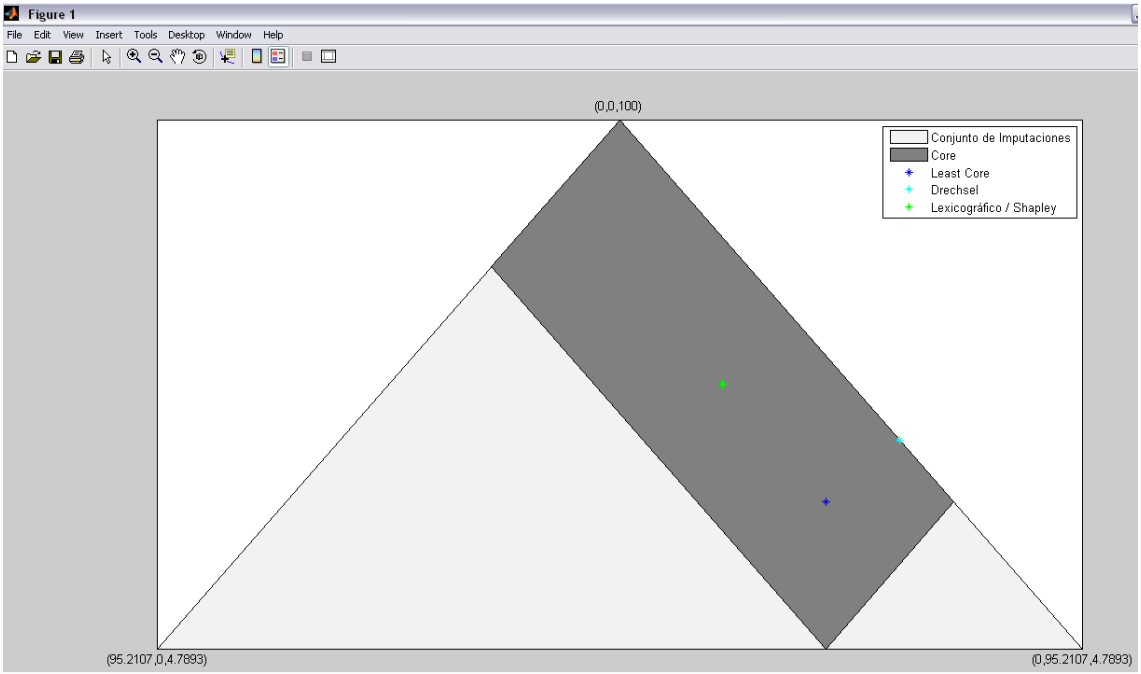


Figura 4-20 Representación gráfica del Core y de las soluciones ofrecidas por los modelos para 3 jugadores

-4 JUGADORES

Este es el código de Matlab que permite obtener gráficamente el Core para 4 jugadores. En primer lugar se declara el vector de la función característica del juego de la bancarrota para 4 jugadores. A continuación , como en la representación para 3 jugadores, se crean los vectores “punto1”, “punto2”, ”punto3” y “punto4” que almacenan el resultado del reparto obtenido con los 4 modelos estudiados. Ahora se representa el Core con la función “Coreset”. Para representar las soluciones obtenidas con los modelos usamos el comando “plot3”.

```
MATLAB 7.5.0 (R2007b)
File  Edit  Debug  Distributed  Desktop  Window  Help
Current Directory: C:\TUGlab
Command Window
b4 =

  Columns 1 through 10

         0         0         0         0   31.8288         0         0   23.6047    4.4784         0

  Columns 11 through 15

   75.4775   56.3512   20.0441   48.1271  100.0000

>> punto1

punto1 =

   25.9364   39.9780   21.8244   12.2612

>> punto2

punto2 =

   35.4996   39.9780   12.2612   12.2612

>> punto3

punto3 =

   36.2365   38.4231   25.3404         0

>> punto4
```

```
punto4 =

   39.6117   35.8659   12.2612   12.2612

>> imputationset(b4),hold on,axis(axis)
>> coreset(b4)
>> plot3(punto1(1),punto1(2),punto1(3),'*b')
>> plot3(punto2(1),punto2(2),punto2(3),'*c')
>> plot3(punto3(1),punto3(2),punto3(3),'*g')
>> plot3(punto4(1),punto4(2),punto4(3),'*m')
```

Figura 4-21 Programa para representar el Core para 4 jugadores

A continuación se muestra la gráfica que representa el Core. Para 4 jugadores viene representado por una figura geométrica en 3 dimensiones. Todos los puntos comprendidos en el interior de este poliedro cumplen los principios de eficiencia y racionalidad individual. Puede observarse que los 4 puntos que representan las soluciones para los modelos estudiados están dentro del Core.

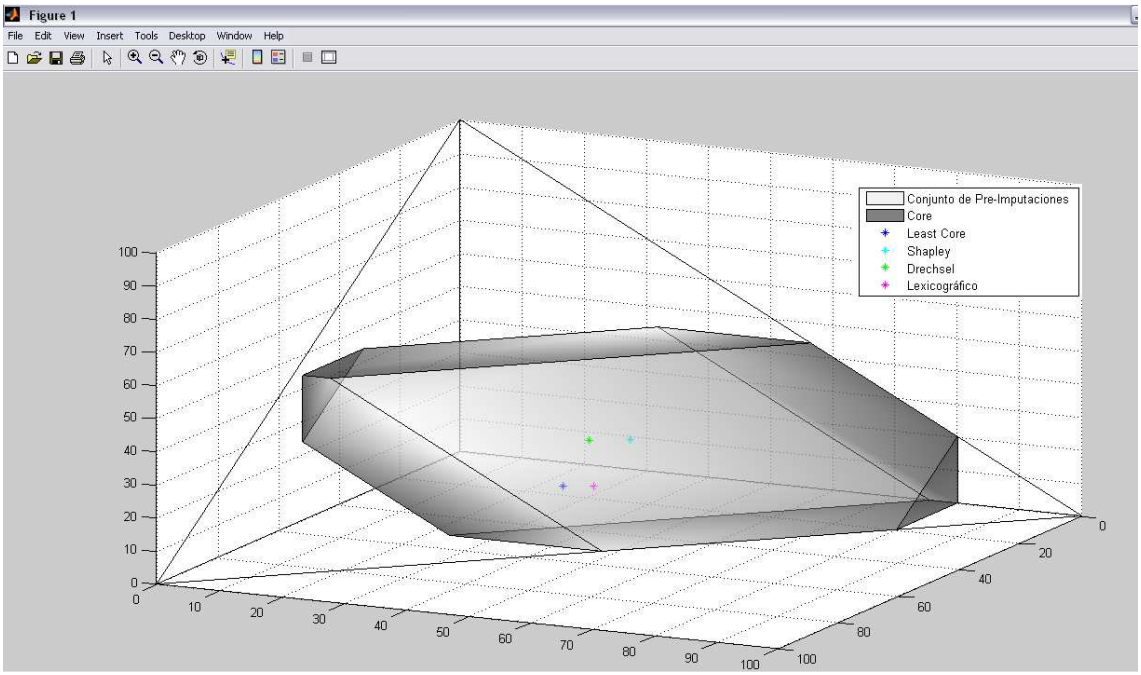


Figura 4-22 Representación gráfica del Core y de las soluciones ofrecidas por los modelos para 4 jugadores

4.2.3 Vector de excesos

-Juego de beneficios

Este algoritmo recibe el vector “x” (solución del juego) y el vector “v” (vector de la función característica) y devuelve el vector “v” cuyas compontes son los excesos, es decir, la diferencia entre el valor de la componente del vector función caracterítica menos el sumatorio del producto de la solución “x” por la columna de la matriz “A” (Matriz de Pertenencia a Coaliciones) correspondiente a la componente que se esté recorriendo.

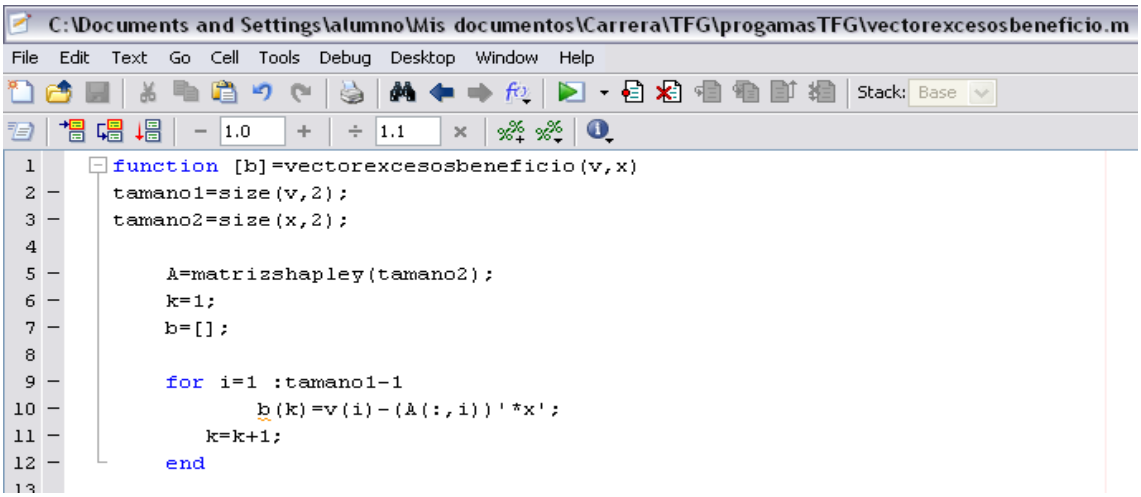
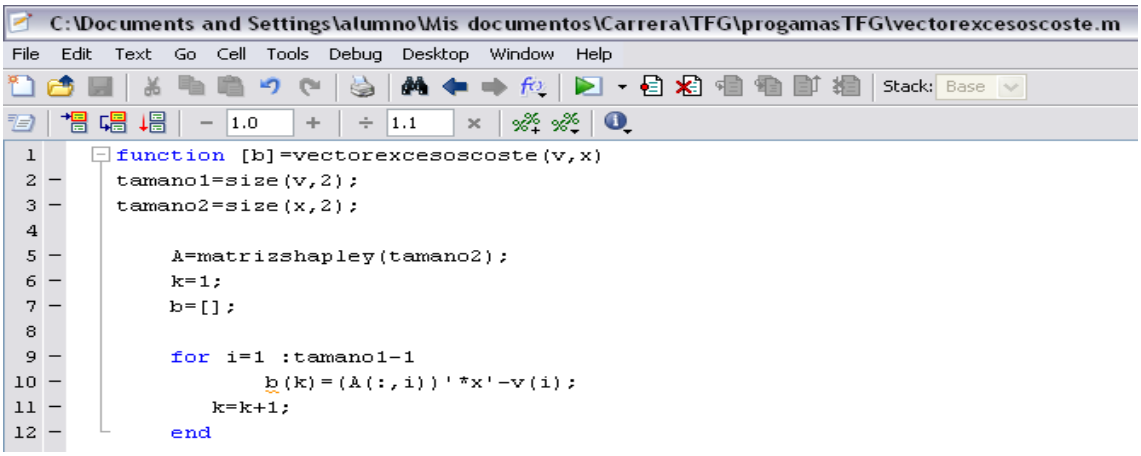


Figura 4-23 Programa vector de excesos (Juegos beneficio)

-Juego de costes

Este algoritmo recibe el vector “x” (solución del juego) y el vector “v “(vector de la función característica) y devuelve el vector “v” cuyas compontes son los excesos es decir, la diferencia entre el sumatorio del producto de la solución” x” por la columna de la matriz “A” (Matriz de Pertenencia a Coaliciones) correspondiente a la componente que se esté recorriendo menos el valor de la componente del vector de la función característica.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES



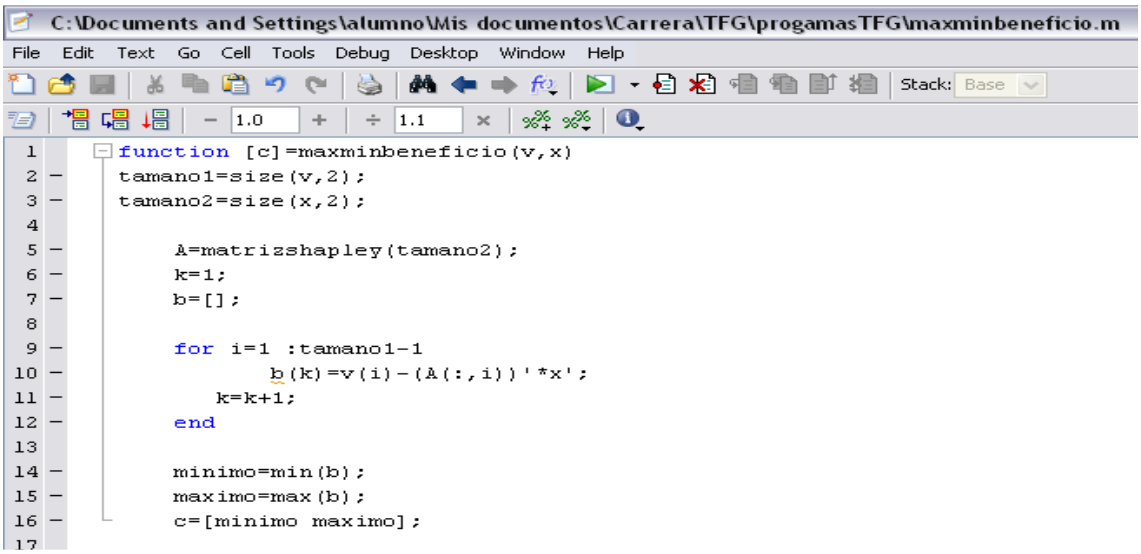

```
function [b]=vectorexcesoscoste(v,x)
tamano1=size(v,2);
tamano2=size(x,2);

     A=matrizshapley(tamano2);
     k=1;
     b=[];

     for i=1 :tamano1-1
            b(k)=(A(:,i))'*x'-v(i);
        k=k+1;
     end
```

Figura 4-24 Programa vector de excesos (Juego coste)

4.2.4 Programa que calcula el máximo y el mínimo de un vector de excesos

-Juego de beneficio

Esta rutina calcula el vector de excesos y sus valores máximos y mínimos los devuelve en el vector "c". Este programa recibe el vector de la función característica y el vector "x" solución del modelo y calcula el máximo y mínimo exceso. Posteriormente se representaran gráficamente los excesos máximos y mínimos calculados por este programa para los 4 modelos y para n=3 hasta n=10 jugadores.




```
function [c]=maxminbeneficio(v,x)
tamano1=size(v,2);
tamano2=size(x,2);

     A=matrizshapley(tamano2);
     k=1;
     b=[];

     for i=1 :tamano1-1
            b(k)=v(i)-(A(:,i))'*x';
        k=k+1;
     end

     minimo=min(b);
     maximo=max(b);
     c=[minimo maximo];
```

Figura 4-25 Programa que calcula Emax y Emin (Juegos beneficio)

-Juego de coste

En este caso el vector de excesos se devuelve como la diferencia entre $\sum_{i \in S} x_i$ - V(S) . Mediante la orden min y max calcula el mínimo y el máximo que son devueltos por el vector "c".

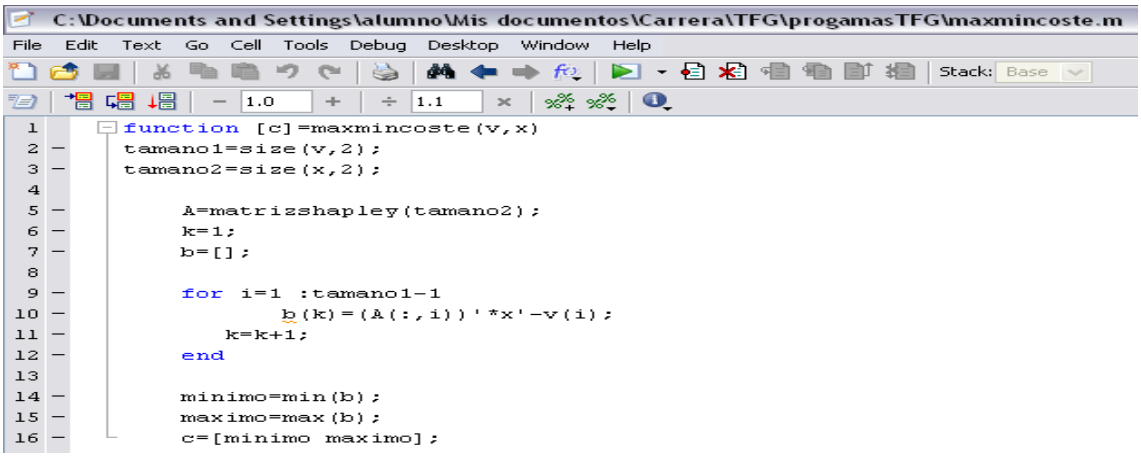


Figura 4-26 Programa que calcula Emax y Emin (Juego coste)

4.3 REPRESENTACIÓN Y COMPARACIÓN DE SOLUCIONES Emax y Emin

4.3.1 Representación numérica y comparación de resultados

A través de las funciones de comprobación se han obtenido las tablas con los resultados de Emax y Emin. A las funciones de comprobación se les ha pasado como parámetro de entrada la solución "x" que ofrecen los distintos modelos.

Como puede observarse, los resultados que ofrecen los modelos Lingo para Least Core (Emax) y para el Lexicográfico (Emin) son los mismos que los calculados a través de las funciones de comprobación, es decir, los valores Emax y Emin calculados a través de las rutinas Matlab explicadas anteriormente, a las que se les ha pasado como parámetros de entrada las soluciones obtenidas con los modelos Lingo.

En ambas tablas se muestran los resultados para 3 hasta 10 jugadores.

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

	Emin							
	3	4	5	6	7	8	9	10
LEAST_CORE	-68,8388	-51,8729	-80,362	-75,3848	-73,0905	-89,3491	-94,1752	-87,5895
DRECHSEL	-57,6776	-61,5769	-69,3922	-77,9422	-85,7617	-86,6751	-97,2083	-83,2943
LEXICOGRÁFICO	-47,6053	-47,7608	-49,7917	-48,7097	-49,993	-49,9554	-49,999	-49,9809
SHAPLEY	-47,6053	-47,7608	-49,7917	-48,7097	-49,993	-49,9554	-49,999	-49,9809

	Emax							
	3	4	5	6	7	8	9	10
LEAST_CORE	-13,1859	-12,2612	-6,54601	-4,74584	-6,19372	-2,13018	-0,46066	-2,25191
DRECHSEL	0	0	0	0	0	0	0	0
LEXICOGRÁFICO	-13,1859	-12,2612	-6,54601	-4,74584	-6,19372	-2,13018	-0,46066	-2,25191
SHAPLEY	-13,1859	-12,2612	-6,54601	-4,74584	-6,19372	-2,13018	-0,46066	-2,25191

	RESULTADOS LINGO							
	3	4	5	6	7	8	9	10
LEAST_CORE (Emax)	-13,1859	-12,2612	-6,54601	-4,74584	-6,19372	-2,13018	-0,46066	-2,25191
LEXICOGRÁFICO (Emin)	-47,6053	-47,7608	-49,7917	-48,7097	-49,993	-49,9554	-49,999	-49,9809

Figura 4-27 Comparativa resultados Emin y Emax obtenidos por los modelos para 3 hasta 10 jugadores

4.3.2 Representación gráfica

A continuación se muestran gráficamente los excesos máximos y mínimos obtenidos para los cuatro modelos estudiados.

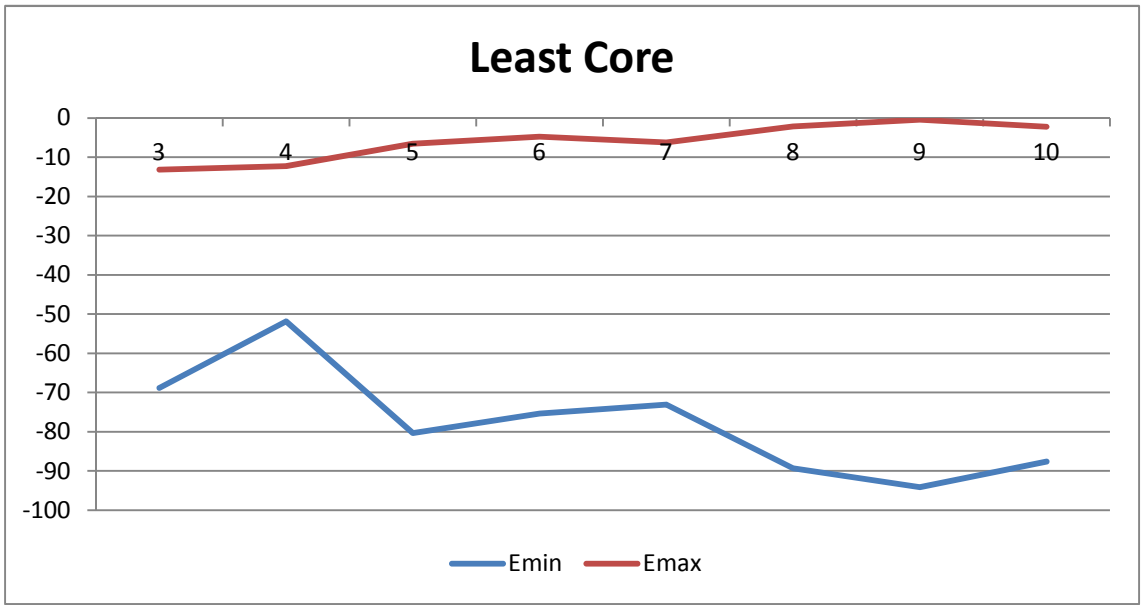


Figura 4-28 Gráfica de resultados de excesos para Least Core

4. EXPERIMENTOS PARA 3 HASTA 10 JUGADORES

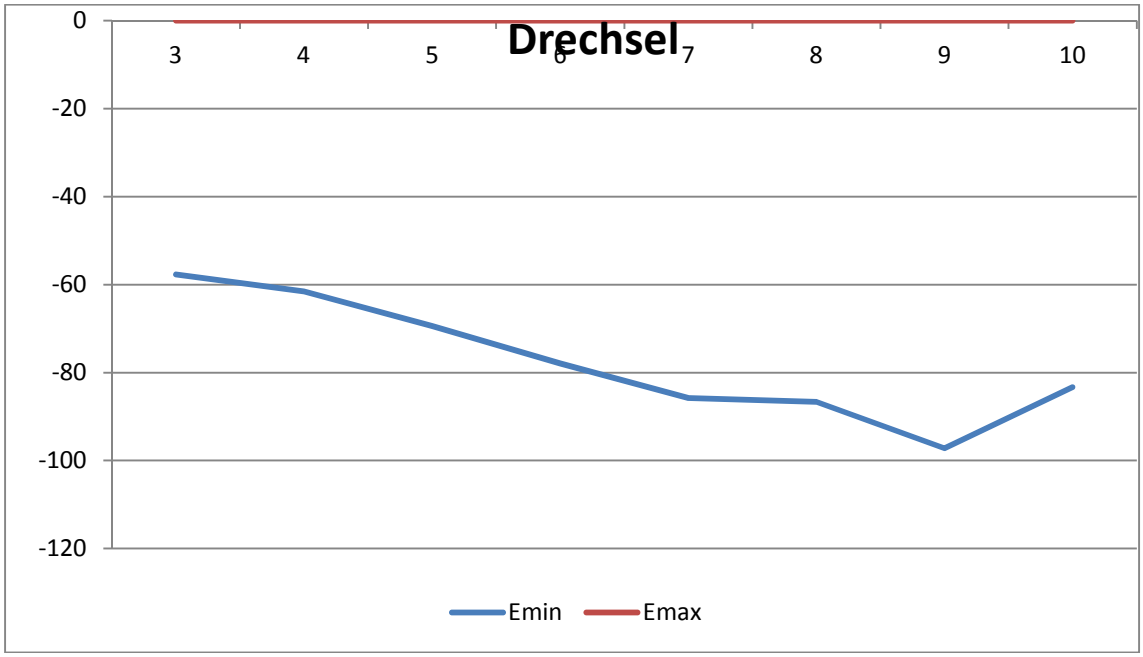


Figura 4-29 Gráfica de resultados de excesos para Drechsel

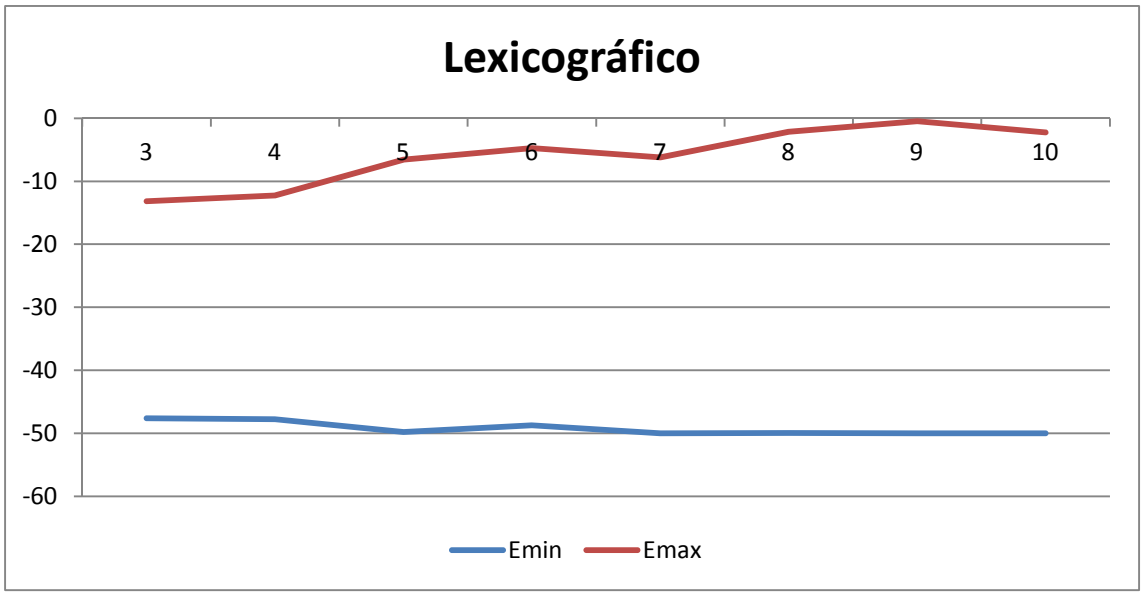


Figura 4-30 Gráfica de resultados de excesos para el Modelo Lexicográfico

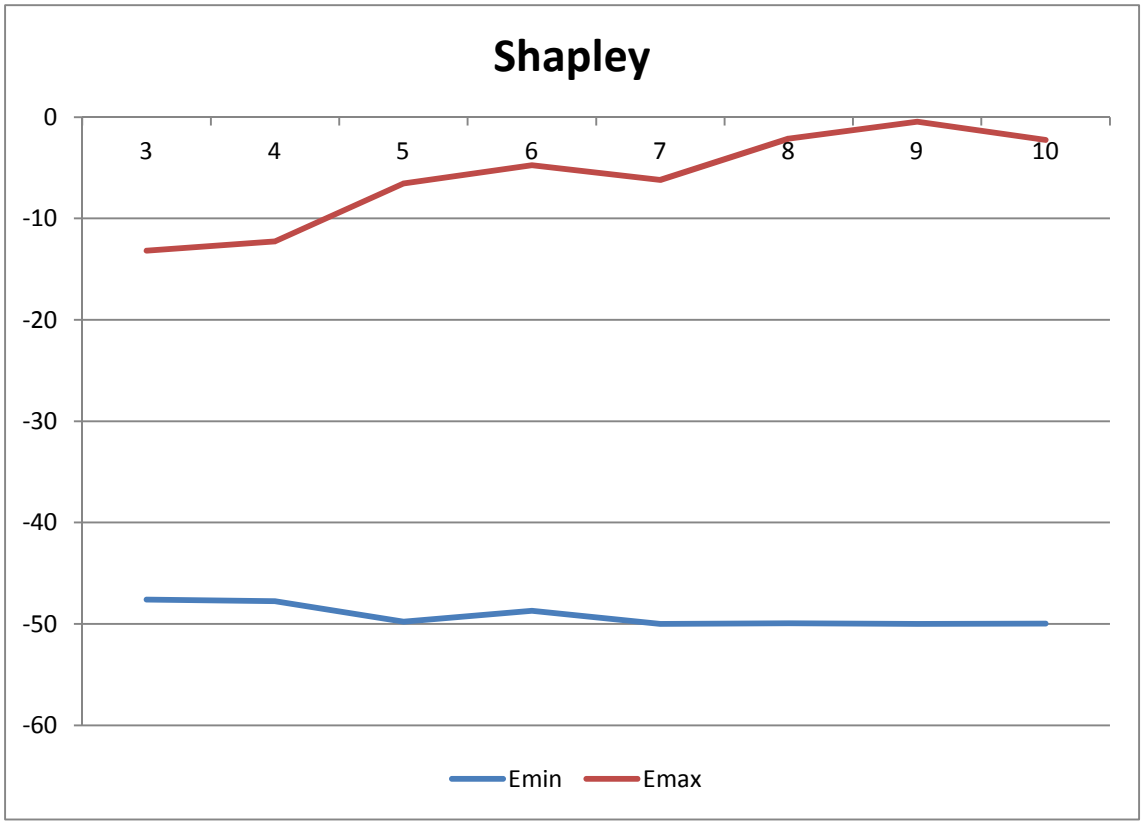


Figura 4-31 Gráfica de resultados de excesos para modelo Shapley

Puede observarse que el modelo Shapley y el modelo Lexicográfico ofrecen soluciones más estables que los otros dos modelos. Es decir, en estos modelos la diferencia entre el exceso máximo y el mínimo tanto para Emax y Emin es mucho menor que en los otros dos modelos. Esto significa que el reparto es más equilibrado y la solución es más equitativa para un juego cooperativo.

5. CONCLUSIONES

Se han realizados los experimentos con el modelo de la bancarrota. Se ha detallado como se crea la función característica para n =3 hasta n =10 jugadores. Para poder aplicar los distintos modelos se ha creado la Matriz de Pertenencia a Coaliciones, que permite establecer las desigualdades para los modelos Least Core, Drechsel y Lexicográfico.

Para el método del valor de Shapley ha sido necesario crear la matriz de Shapley a través del programa Matlab. Esta matriz de Shapley se multiplica por el vector de la función caracterísa y da directamente la solución.

Puede apreciarse que los modelos “Lexicográfico” y “ Valor de Shapley “ ofrecen soluciones más equilibradas tanto para el cálculo de los repartos como para los excesos máximos y mínimos. Es decir, aporta repartos más equitativos para el conjunto de los jugadores, lo cual es el objeto de los juegos cooperativos.

6. REFERENCIAS / BIBLIOGRAFÍA

Referencias Core:

Gillies, D.B., Some theorems on n-person games, PhD Thesis, Princeton University Press, Princeton, NJ, 1953.

Gillies, D.B., 1959. Solutions to general non-zero-sum games. In: A.W. Tucker and R.D. Luce, eds. Contributions to the Theory of Games IV. Princeton: Princeton University Press, 47–85.

Referencia Least Core:

Maschler, M., Peleg, B., and Shapley, L.S., 1979. Geometric properties of the kernel, nucleolus, and related solution concepts. Mathematics of Operations Research, 4, 303–338.

Referencia Valor de Shapley:

Shapley, L.S., The value of an n-person game, in: H.W. Kuhn, A.W. Tucker (Eds.), Contributions to the Theory of Games, Princeton University Press, 1953

Referencia Minmax core (Drechsel):

Drechsel, J. and Kimms, A.(2011) 'Cooperative lot sizing with transshipments and scarce capacities: solutions and fair cost allocations', International Journal of Production Research, 49: 9, 2643 — 2668

Referencias generales:

Myerson, R.B., 1997. Game Theory – Analysis of Conflict. Cambridge: Harvard University Press.

Osborne, M.J. and Rubinstein, A., 1994. A Course in Game Theory. Cambridge: The MIT Press.

Owen, G., 2001. Game Theory. 3rd ed. San Diego: Academic Press.

Peleg, B. and Sudhölter, P., 2003. Introduction to the Theory of Cooperative Games. 3rd ed. Boston: Kluwer

Lozano, S., Apuntes de Teoría de Juegos Cooperativos, Asignatura Métodos Cuantitativos Avanzados de Gestión, 2017

7. ANEXOS

7.1 PROGRAMA PARA OBTENER LOS ÍNDICES DEL VECTOR CARACTERÍSTICO

Este programa permite obtener un vector en el que cada componente representa los jugadores que participan en cada coalición.
En primer lugar hay que definir n, a continuación llama a la función de Shapley de n. Ahora entra en un bucle en el que crea un vector que es el valor de la columna de la matriz A que se está recorriendo. A continuación recorre el vector "a" y si se encuentra un 1 almacena en el vector en formato carácter el valor de la fila que está recorriendo y el signo ",". Sigue este procedimiento hasta recorrer todas las columnas.
Al final define la primera componente del vector resultante como "{" y cambia la última componente por "}". Pues el carácter "," asociado a la última componente sobra.
Mediante el comando xlswrite copia en una hoja excel el vector resultante.

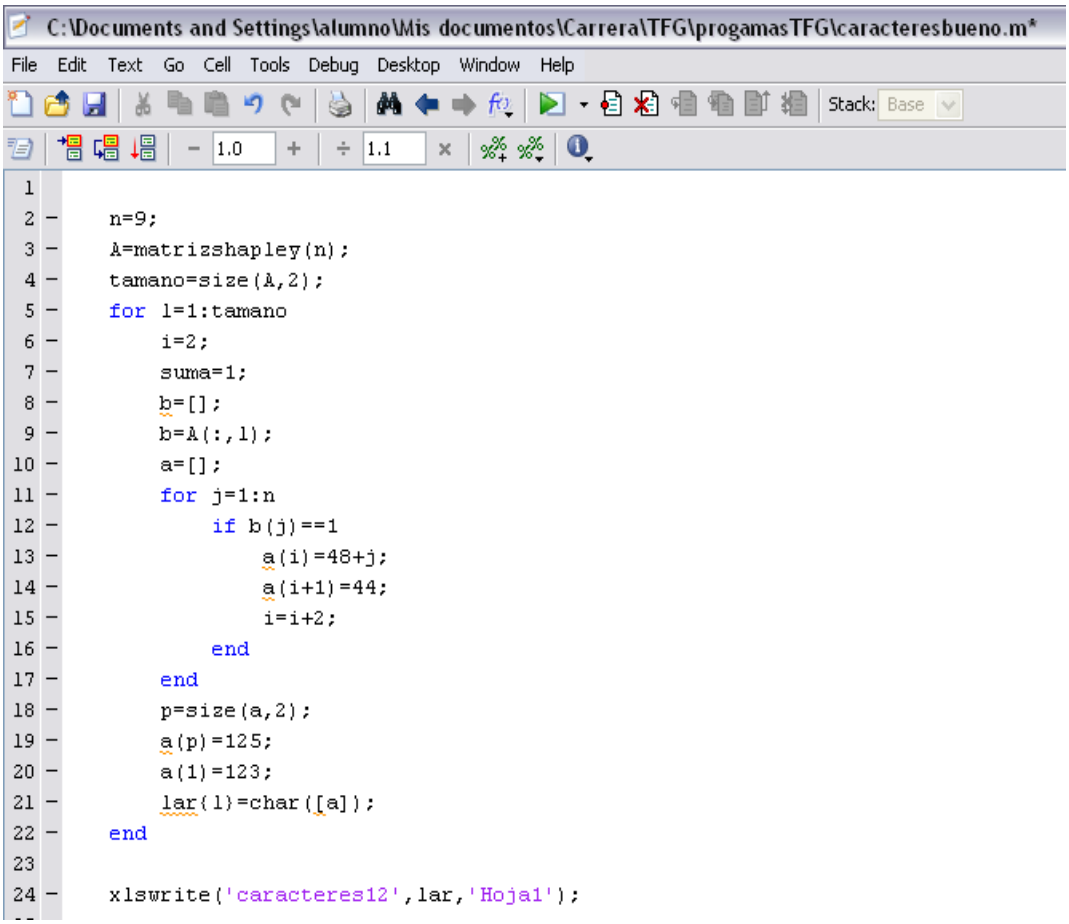


Figura 7-1 Programa que calcula los índices del vector de una función característica

Ejemplo para 3 jugadores:

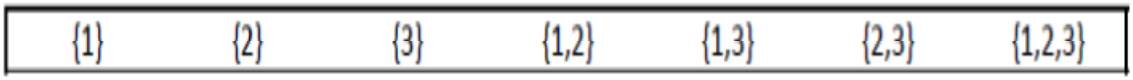


Figura 7-2 Vector de jugadores que intervienen en una coalición para 3 jugadores

7.2 PROGRAMAS PARA OBTENER Emax, Emin

7.2.1 Least Core

Esta función lee la matriz A (resultados del algoritmo Least Core para 3 hasta 10 jugadores) y la matriz B (matriz de 8 filas en la que cada fila representa las componentes del vector de la función característica para 3 hasta 10 jugadores) ambas guardadas en un archivo Excel. A continuación guarda las filas de ambas matrices como vectores (a y b), luego aplica la función maxmin a cada par de vectores para obtener los excesos máximos y mínimos y por último guarda estos resultados en el vector D.

C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\progamasTFG\Comprobacion funciones\vectorleastcore.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

```
A=xlsread('resultadosleastcore.xlsx');
B=xlsread('matrizvectorcaracteristico.xlsx');
a3=[];b3=[];a4=[];b4=[];a5=[];b5=[];a6=[];b6=[];
a7=[];b7=[];a8=[];b8=[];a9=[];b9=[];a10=[];b10=[];
c3=[];c4=[];c5=[];c6=[];c7=[];c8=[];c9=[];c10=[];D=[];


for i=1:3
    a3(i)=A(1,i);
end
for i=1:4
    a4(i)=A(2,i);
end
for i=1:5
    a5(i)=A(3,i);
end
for i=1:6
    a6(i)=A(4,i);
end
for i=1:7
    a7(i)=A(5,i);
end
for i=1:8
    a8(i)=A(6,i);
end
for i=1:9
    a9(i)=A(7,i);
end
for i=1:10
    a10(i)=A(8,i);
end


for i=1:7
    b3(i)=B(1,i);
end
```

7. ANEXOS

```
37 -    for i=1:15
38 -        b4(i)=B(2,i);
39 -    end
40 -    for i=1:31
41 -        b5(i)=B(3,i);
42 -    end
43 -    for i=1:63
44 -        b6(i)=B(4,i);
45 -    end
46 -    for i=1:127
47 -        b7(i)=B(5,i);
48 -    end
49 -    for i=1:255
50 -        b8(i)=B(6,i);
51 -    end
52 -    for i=1:511
53 -        b9(i)=B(7,i);
54 -    end
55 -    for i=1:1023
56 -        b10(i)=B(8,i);
57 -    end
58
59 -    c3=maxmin(b3,a3);
60 -    c4=maxmin(b4,a4);
61 -    c5=maxmin(b5,a5);
62 -    c6=maxmin(b6,a6);
63 -    c7=maxmin(b7,a7);
64 -    c8=maxmin(b8,a8);
65 -    c9=maxmin(b9,a9);
66 -    c10=maxmin(b10,a10);
67
68 -    D=[c3;c4;c5;c6;c7;c8;c9;c10];
69
70 -    xlswrite('leastcore',D,'Hoja1','A1');
```

Figura 7-3 Programa que calcula Emax y Emin (Least Core)

El procedimiento para calcular los excesos para los demás modelos es similar.

7.2.2 Drechsel

C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\progamasTFG\Comprobacion funciones\vectordrechsel.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

```
 1 -    A=xlsread('resultadosdrechsel.xlsx');
 2 -    B=xlsread('matrizvectorcaracteristico.xlsx');
 3 -    a3=[];b3=[];a4=[];b4=[];a5=[];b5=[];a6=[];b6=[];
 4 -    a7=[];b7=[];a8=[];b8=[];a9=[];b9=[];a10=[];b10=[];
 5 -    c3=[];c4=[];c5=[];c6=[];c7=[];c8=[];c9=[];c10=[];D=[];
 6
 7 -    for i=1:3
 8 -        a3(i)=A(1,i);
 9 -    end
10 -    for i=1:4
11 -        a4(i)=A(2,i);
12 -    end
13 -    for i=1:5
14 -        a5(i)=A(3,i);
15 -    end
16 -    for i=1:6
17 -        a6(i)=A(4,i);
18 -    end
19 -    for i=1:7
20 -        a7(i)=A(5,i);
21 -    end
22 -    for i=1:8
23 -        a8(i)=A(6,i);
24 -    end
25 -    for i=1:9
26 -        a9(i)=A(7,i);
27 -    end
28 -    for i=1:10
29 -        a10(i)=A(8,i);
30 -    end
31
32 -    for i=1:7
33 -        b3(i)=B(1,i);
34 -    end
35 -    for i=1:15
36 -        b4(i)=B(2,i);
```

7. ANEXOS

```
    end
    for i=1:31
        b5(i)=B(3,i);
    end
    for i=1:63
        b6(i)=B(4,i);
    end
    for i=1:127
        b7(i)=B(5,i);
    end
    for i=1:255
        b8(i)=B(6,i);
    end
    for i=1:511
        b9(i)=B(7,i);
    end
    for i=1:1023
        b10(i)=B(8,i);
    end


    c3=maxmindrechsel(b3,a3);
    c4=maxmindrechsel(b4,a4);
    c5=maxmindrechsel(b5,a5);
    c6=maxmindrechsel(b6,a6);
    c7=maxmindrechsel(b7,a7);
    c8=maxmindrechsel(b8,a8);
    c9=maxmindrechsel(b9,a9);
    c10=maxmindrechsel(b10,a10);


    D=[c3;c4;c5;c6;c7;c8;c9;c10];

    xlswrite('drechsel',D,'Hoja1','A1');
```

Figura 7-4 Programa que calcula Emax y Emin (Drechsel)

7.2.3 Lexicográfico

C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\progamasTFG\Comprobacion funciones\vectorlexicografico.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
    A=xlsread('resultadoslexicografico.xlsx');
    B=xlsread('matrisvectorcaracteristico.xlsx');
    a3=[];b3=[];a4=[];b4=[];a5=[];b5=[];a6=[];b6=[];
    a7=[];b7=[];a8=[];b8=[];a9=[];b9=[];a10=[];b10=[];
    c3=[];c4=[];c5=[];c6=[];c7=[];c8=[];c9=[];c10=[];D=[];

    for i=1:3
        a3(i)=A(1,i);
    end
    for i=1:4
        a4(i)=A(2,i);
    end
    for i=1:5
        a5(i)=A(3,i);
    end
    for i=1:6
        a6(i)=A(4,i);
    end
    for i=1:7
        a7(i)=A(5,i);
    end
    for i=1:8
        a8(i)=A(6,i);
    end
    for i=1:9
        a9(i)=A(7,i);
    end
    for i=1:10
        a10(i)=A(8,i);
    end


    for i=1:7
        b3(i)=B(1,i);
    end
    for i=1:15
```

7. ANEXOS

```
        b4(i)=B(2,i);
    end
    for i=1:31
        b5(i)=B(3,i);
    end
    for i=1:63
        b6(i)=B(4,i);
    end
    for i=1:127
        b7(i)=B(5,i);
    end
    for i=1:255
        b8(i)=B(6,i);
    end
    for i=1:511
        b9(i)=B(7,i);
    end
    for i=1:1023
        b10(i)=B(8,i);
    end

    c3=maxmin(b3,a3);
    c4=maxmin(b4,a4);
    c5=maxmin(b5,a5);
    c6=maxmin(b6,a6);
    c7=maxmin(b7,a7);
    c8=maxmin(b8,a8);
    c9=maxmin(b9,a9);
    c10=maxmin(b10,a10);

    D=[c3;c4;c5;c6;c7;c8;c9;c10];

    xlswrite('lexicografico',D,'Hoja1','A1');
```

Figura 7-5 Programa que calcula Emax y Emin (Lexicográfico)

7.2.4 Shapley

C:\Documents and Settings\alumno\Mis documentos\Carrera\TFG\progamasTFG\Comprobacion funciones\vectorshapley.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

```
    A=xlsread('resultadosshapley.xlsx');
    B=xlsread('matrizvectorcaracteristico.xlsx');
    a3=[];b3=[];a4=[];b4=[];a5=[];b5=[];a6=[];b6=[];
    a7=[];b7=[];a8=[];b8=[];a9=[];b9=[];a10=[];b10=[];
    c3=[];c4=[];c5=[];c6=[];c7=[];c8=[];c9=[];c10=[];D=[];

    for i=1:3
        a3(i)=A(1,i);
    end
    for i=1:4
        a4(i)=A(2,i);
    end
    for i=1:5
        a5(i)=A(3,i);
    end
    for i=1:6
        a6(i)=A(4,i);
    end
    for i=1:7
        a7(i)=A(5,i);
    end
    for i=1:8
        a8(i)=A(6,i);
    end
    for i=1:9
        a9(i)=A(7,i);
    end
    for i=1:10
        a10(i)=A(8,i);
    end

    for i=1:7
        b3(i)=B(1,i);
    end
    for i=1:15
        b4(i)=B(2,i);
```

7. ANEXOS

```
    end
    for i=1:31
        b5(i)=B(3,i);
    end
    for i=1:63
        b6(i)=B(4,i);
    end
    for i=1:127
        b7(i)=B(5,i);
    end
    for i=1:255
        b8(i)=B(6,i);
    end
    for i=1:511
        b9(i)=B(7,i);
    end
    for i=1:1023
        b10(i)=B(8,i);
    end

    c3=maxmin(b3,a3);
    c4=maxmin(b4,a4);
    c5=maxmin(b5,a5);
    c6=maxmin(b6,a6);
    c7=maxmin(b7,a7);
    c8=maxmin(b8,a8);
    c9=maxmin(b9,a9);
    c10=maxmin(b10,a10);

    D=[c3;c4;c5;c6;c7;c8;c9;c10];

    xlswrite('shapley',D,'Hoja1','A1');
```

Figura 7-6 Programa que calcula Emax y Emin (Shapley)