Universidad Nacional Autónoma de México

Facultad de Ciencias

INTRODUCCIÓN A LA TEORÍA DE JUEGOS NO COOPERATIVOS RECTANGULARES FINITOS CON R

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

ACTUARIO

PRESENTA:

HUGO IVÁN PORTILLA RAMÍREZ

TUTORA

ACT. CLAUDIA VILLEGAS AZCORRA

2019

1. Datos del alumno
Portilla
Ramírez
Hugo Iván
55 32 51 07 49
Universidad Nacional Autónoma de México
Facultad de Ciencias
Actuaría
108001798

2. Datos de la tutora
Act.
Claudia
Villegas
Azcorra

3. Datos del sinodal 1
Mae.
Juan Carlos
Guapilla
Salamanca

4. Datos del sinodal 2
Dr.
Antonio
Carrillo
Ledesma

5. Datos del sinodal 3
M. en I.
Minerva Elizabeth
Soto
Patiño

6. Datos del sinodal 4
Act.
Alejandro Felipe
Zarate
Pérez

7. Datos del trabajo escrito
Introducción a la teoría de juegos no cooperativos rectangulares finitos con R.
162 p
2019

Agradecimientos

Somos la suma de los esfuerzos que otros han hecho por nosotros, por ello, esta tesis no es sólo mía sino también de las increíbles personas que me han acompañado en los diferentes retos y decisiones que me han convertido en la persona que soy.

Primero que nadie agradezco a mi mamá, mi papá, mi hermana, y mi primo Diego su infinita paciencia, enseñanza y amor. Sin ellos quién sabe que tan lóbrega sería mi persona.

También, esta tesis se debe a los amigos que día a día me inspiran, acompañan y demuestran que el mundo esta lleno de personas extraordinarias. En orden de aparición en mi vida: Nancy, Carlos, Diego, Iván, Citlali, Ingrid, Karen, "las believers" , Chío, Rodolfo, Omar, Allison, Pao, y "los cajeritos".

Agradezco a la universidad y su comunidad, que ha sido mi hogar desde mi formación secundaria hasta ahora, por haberme dado la oportunidad de formar parte activa de un maravilloso espacio de transmisión del conocimiento y debate de ideas. Me encuentro infinitamente en deuda con cada uno de mis profesores y alumnos.

Hago un reconocimiento especial a mentores que me orientaron en momentos clave. En orden cronológico: el profesor Alfredo Montiel, el profesor Fernando Quintana, el licenciado Antonio Mosso, la profesora Claudia Villegas quien no obstante con orientarme en la vida también tuvo el invaluable esmero de dirigir esta tesis, y por supuesto, a mis sinodales: el maestro Juan Carlos, el doctor Antonio Carrillo, la maestra Minerva Soto y el pro-

fesor Zárate, quienes destinaron mucho recurso de tiempo y conocimiento para aportar a mi formación profesional.

¡Gracias!

Al lector

El conocimiento crece cuando se comparte.

Tengo la intención de mantener en el futuro versiones actualizadas de los contenidos de este texto, con el objetivo de compartirlos con todo aquel lector que se encuentre interesado.

Este material no existiría sin las anotaciones, preguntas y sugerencias de mentores, compañeros y alumnos.

Cualquier comentario o aporte acerca de este trabajo será bien recibido a través de los siguientes medios de contacto.

hugoportillar@ciencias.unam.mx
https://github.com/hugoportillar22/GameTheory

Algunas veces nos toca enseñar pero siempre nos toca aprender.

Refrán popular

Índice general

Introducción

Contexto

El presente material muestra una introducción, con herramientas que hacen uso de las tecnologías de la información y cómputo, a la teoría de juegos. Lo anterior se encuentra apegado al curso de "Teoría de Juegos en Economía" que se imparte en la Facultad de Ciencias y cuyo objetivo es comprender y ser capaz de utilizar, en términos matemáticos, los conceptos básicos de la teoría de juegos para el análisis de conflictos sociales y económicos.

La materia, al tener una vasta aplicación y algoritmos notoriamente delimitados, ofrece de manera ideal la posibilidad de abordarse con las herramientas que la programación ofrece. Los entornos de programación pueden promover el uso de la teoría de juegos en más contextos actuariales, ya que, al permitir la rápida, directa y económica exploración y experimentación del conocimiento matemático, más alumnos y profesionales pueden encontrar aplicaciones en sus diversas áreas y así, contribuir a la creación de mayor conocimiento (Véase [C13]).

La tecnología abre espacios para que el estudiante pueda vivir nuevas experiencias matemáticas difíciles de lograr en medios tradicionales como el lápiz y el papel.

Lo anterior puede resultar de particular interés ya que la teoría de juegos tiene una notable aplicación práctica en el estudio de problemáticas relacionadas con la búsqueda de posibles resultados de conflictos entre individuos, tales como guerras, competencia económica, selección natural, etcétera (Véase [A3]).

Por ejemplo, John Von Neumann en 1944, cuando publicó su libro "Teoría

de Juegos y Comportamiento Económico", concibió a la Teoría de Juegos como la matemática ad hoc para estudiar la problemática económica, pues consideraba que el enfoque usual dentro de la teoría de su época no correspondía a la realidad de los conflictos económicos. Este enfoque de la teoría económica situaba a agentes, individuales y aislados, optimizando sus recursos sin tomar en cuenta las posibilidades de acción que tenían los demás individuos (Véase [A5]).

Precisamente, John Von Neumann es considerado uno de los padres de la computadora moderna ya que contribuyó enormemente a su desarrollo y posibilitó en la generación de computadoras cuyos programas se almacenaran en memoria (Véase [B7]).

Problema a resolver

Un curso de teoría de juegos podría ser enriquecido si se abordara de manera exitosa con el apoyo de herramientas de programación, sin embargo, la problemática actual es que resulta muy poco frecuente encontrar cursos que vayan de la mano con dichas herramientas debido, entre otras causas, al escaso e inadecuado material de programación que se encuentra disponible.

A pesar de existir un amplio desarrollo de herramientas de programación, para el área de teoría de juegos solo se pueden encontrar algunas paqueterías y programas de ciertos tópicos que suelen resolver problemas muy particulares y con un sustento teórico concreto o avanzado. Además, cada proyecto esta escrito con diferentes formalidades, paradigmas y lenguajes de programación según lo hayan preferido o decidido sus autores.

Objetivo general

Aportar a la docencia didáctica de los principales conceptos y técnicas de los juegos no cooperativos rectangulares finitos con el uso de las tecnologías de la información y cómputo, a través del desarrollo y presentación de un conjunto de códigos que permitan experimentar y dar solución ágil a problemas asociados a la materia que, por la vía

tradicional, resultan muy laboriosos. Tales códigos están a disposición del lector en el siguiente repositorio Github dedicado al proyecto.

`https://github.com/hugoportillar22/GameTheory`

Objetivos particulares

- Crear códigos en los que se programe cada algoritmo contenido en los temas correspondientes a los juegos no cooperativos rectangulares finitos. Estos códigos deberán ser capaces de resolver los problemas que pueden resolverse con el algoritmo teórico.

- Presentar en este texto de forma didáctica, concisa y fundamentada, la ejemplificación de uso de los programas generados.

- Compilar y garantizar la accesibilidad a los programas, pseudocódigos y ejemplificaciones, en el siguiente repositorio: `https://github.com/hugoportillar22/GameTheory` para su descarga directa o bien, si se tiene instalado Git, para poder descargar todos los fuentes usando:

```
1
2 git clone git://github.com/hugoportillar22/GameTheory
3
```

- Generar interés en cualquier persona que encuentre útiles las herramientas de programación descritas en este texto exhortándolos a seguir experimentando con los códigos.

Metodología

Para el desarrollo del trabajo se exponen ejemplos de casos de uso resueltos tanto de manera teórica, como con las herramientas computacionales de los cinco temas básicos del estudio de juegos no cooperativos finitos rectangulares que abordan los textos de referencia de la materia (Véase [A3]).

Las demostraciones provenientes de la teoría, así como el uso y manejo de R no serán profundizadas en el presente proyecto ya que no forman parte

del alcance del mismo, asimismo se enlistan en el apéndice “A.Algunas nociones de programación” y en la bibliografía distintos recursos que pueden ser utilizados para dicho fin.

Atendiendo a la premisa de incentivar el interés de todo aquel que pueda resultarle útil este trabajo, el código desarrollado sigue buenas practicas de condificación con la finalidad de tener una alta calidad didáctica.

Los contenidos se organizan de la siguiente manera: el primer capítulo corresponde al marco teórico, donde se fundamentan los conceptos necesarios de la teoría de juegos y del entorno de programación. Así mismo se argumenta la relevancia, beneficios, la elección de los temas a desarrollar y del software a utilizar. También se establecen algunos códigos y convenciones que serán relevantes más adelante.

Los capítulos del 2 al 6 presentan individualmente los 6 tópicos algorítmicos comprendidos en la delimitación de este trabajo que son en orden: Dominancias, equilibrio de Nash, estrategias conservadoras y máximo asegurable, antagonismo y punto silla. Todas estas unidades se componen de varios subtemas, cada uno relativo a una forma o punto de inicio diferente para efectuar el algoritmo asociado al tema.

Es importante mencionar que, en la mayoría de los casos, cada unidad temática, se divide en ‘Juegos de estrategias puras’ y ‘Juegos de estrategias mixtas’. Las estrategias mixtas son una forma que tendrán los jugadores para combinar, en particular, a través del tiempo, sus estrategias puras, por lo cuál, se generalizará, para ellas, mucho de lo que se hace con estrategias puras.

Las secciones de los capítulos temáticos se procura que sean abordadas con la siguiente estructura:

1. Definición: Se establece la motivación de la existencia de la técnica en cuestión y su respectivo fundamento matemático.

2. Contexto de un problema: Se plantea un problema ilustrativo hipotético que puede ser resuelto con lo provisto.

3. Solución del problema en forma teórica: Se resuelve el problema con la teoría o el algoritmo teórico asociado.

4. Solución del problema con programación: Se obtienen soluciones y resultados del problema empleando los códigos de los programas ge-

nerados disponibles en el repositorio de Github, mostrando su capacidad y beneficio.

Capítulo 1

Marco teórico

1.1. Conceptos básicos de Teoría de Juegos

Se entiende por juego a un modelo simplificado de un conflicto en donde cada participante, llamado jugador, debe tomar una o varias decisiones dentro de un conjunto de posibles estrategias o reglas que permitan alcanzar su objetivo, que es tratar de conseguir la mayor utilidad posible. Al final de todo juego hay un resultado determinado: una recompensa o un castigo para cada uno de los participantes (Véase [A5]).

El resultado que se obtiene en dicho conflicto es consecuencia de las decisiones de todos los jugadores y en algunos casos del azar, y puede incluso resultar contrario a lo que cada uno se planteó como objetivo.

Se debe enfatizar que el término "juego" no se restringe a los juegos lúdicos, que son los que se suelen asociar directamente con la palabra. La palabra juego, según el concepto que nos compete, puede abarcar cualquier tipo de situación de interacción social, permitiendo modelar problemas de política, economía o de otras índoles de relaciones humanas (Véase [A2]).

1.1.1. Elementos escenciales de la teoría de juegos

Los elementos esenciales de la teoría de juegos son: el jugador, la función de pago, y la estrategia (Véase [A1]).

- *Jugador*
  Del concepto de juego se sabe que los jugadores son aquellos parti-

cipantes capaces de tomar decisiones que conduzcan a la resolución del juego en el que se encuentren, y para ello se establecen tres características que debe poseer cualquier jugador.

1. Es considerado como "persona racional" , es decir:
   a) Puede prever los estados futuros posibles que dependen de la acción que ha tomado en el presente.
   b) Tiene "metas" o "valores" por medio de los cuales puede evaluar el interés de tales estados futuros.
   c) Toma la acción que le conduce al estado que le despierta mayor interés, con base a sus metas o valores ya que se tiene como supuesto que ningún jugador está motivado por el autodestructivo deseo de perder.
2. Cada jugador tiene un objetivo e interpretación de "resultado justo" de acuerdo a sus preferencias subjetivas, costumbres sociales, dogmas e incluso personalidad.
3. No es necesario que un jugador sea una persona; puede ser un equipo, una corporación, una nación, etc. Sin embargo, es útil considerar como un solo jugador a un grupo de individuos que tienen intereses idénticos con respecto al juego.

- *Función de pago o de pagos*
  Es el conjunto de recompensas o castigos que reciben los jugadores implicados en un juego cuando este llega a su resultado final.

  En general se habla de los resultados de los juegos como conjuntos de números reales que por cada jugador contienen un elemento que expresa su respectivo pago. Esto puede crear la falsa impresión de que el pago que los jugadores reciben es siempre en dinero. La idea no es esa, los pagos o resultados para un jugador pueden ser hechos cualitativos, subjetivos de medir como la felicidad generada o un favor obtenido. Sin embargo, recurriendo a la llamada teoría de la utilidad se pueden convertir en números reales los diversos pagos de los jugadores.

- *Estrategia*
  En situaciones donde se tiene que tomar decisiones cuyos resultados

directos son inciertos, ya sea porque no se puede controlar las decisiones del adversario o porque existen diferentes factores desconocidos o azarosos, cada jugador toma una decisión adecuada de entre un conjunto de posibilidades que pueden suceder. Al conjunto de posibles decisiones elegibles por los jugadores se le llama perfil o conjunto de estrategias. Dicho de otra forma, una estrategia es una descripción completa de como uno se comportará bajo cada una de las posibles circunstancias.

1.1.2. Definición de juego

Una vez explicados los conceptos es oportuno establecer una definición formal de juego. La siguiente definición no es única, pero es la que mejor se apega a la perspectiva general que persigue este trabajo (Véase [A3]).

Definición 1.1.2.1. *Sean:*

- N *el conjunto de jugadores,*
- D_j *el conjunto de estrategias puras del jugador* j*,*
- φ_j *la función de pago del jugador* j*, entonces*

$$(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$$

denotará el juego que tiene el conjunto de jugadores N*, los conjuntos de estrategias puras* D_j *y la funciones de pago* φ_j*.*

Se usará la letra D *para denotar a* $D_1 \times D_2 \times \ldots \times D_n$ *si* N *es finito y tiene* n *elementos, es decir:*

$$D = \prod_{j=1}^{n} D_j, \#N = n$$

a los elementos de este conjunto les llama perfiles de estrategias puras y corresponden a las estrategias elegidas por los jugadores $j \in N$*.*

Los perfiles de estrategias se denotan por la letra σ *como aquellos tales que* $\sigma_k \in D, \quad \#D = k$

1.1.3. Tipos de juegos

Cada juego es único y se puede abordar desde diferentes perspectivas. Aun así, los juegos no cooperativos se pueden clasificar, en líneas generales, según sus características más relevantes (Véase [A3]):

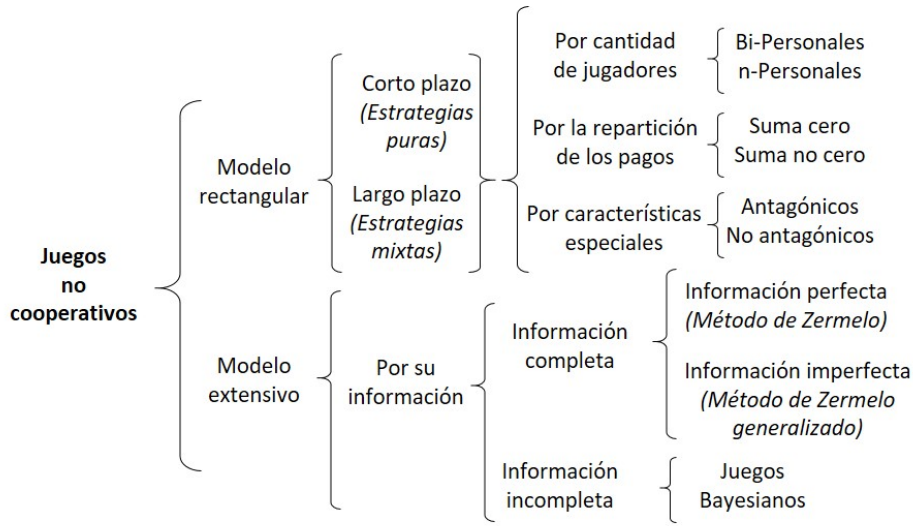


Figura 1.1: Clasificación de los juegos no cooperativos según sus características primordiales.

A continuación se explican, a grandes rasgos, aquellas clases de juegos que delimitan a los temas que se abordaran más adelante.

Juegos finitos

Retomando lo que se presentó formalmente en la definición de juego, un juego finito es aquel que tiene un número finito de perfiles de estrategias. En contraparte, en un juego infinito habrá una cantidad ilimitada de estrategias a tomar (Véase [A4]).

Definición 1.1.3.1. *Si para un juego* $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$*, el conjunto* N *de tamaño* n *de jugadores, y los conjuntos* D_j *son finitos, se dice que el juego es finito, es decir:*

$$Si\ N < \infty \wedge D_j < \infty, \forall j \in N$$

$$\Rightarrow (N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N}) < \infty$$

Juegos cooperativos y no cooperativos

Los *juegos cooperativos* tienen un papel normativo, buscan los resultados "equitativos", "justos" que conseguirían agentes "racionales" y "bien informados". Los diversos conceptos de solución que hay en la teoría que los estudia se establecen con conjuntos de axiomas que responden a una forma de entender esas propiedades de racionalidad, justicia y equidad.

Los *juegos no cooperativos*, en cambio, son un marco teórico adecuado para estudiar si hay una "ley" interna en el conflicto que se estudia, y pueden resultar un importante instrumento de análisis, razón por la que este texto se centra en los juegos no cooperativos (Véase [A1]).

Juegos rectangulares y extensivos

Los dos modelos de juegos que se trabajan en la teoría de los juegos no cooperativos son, por un lado, el rectangular o estratégico que consta, como hemos planteado antes, del conjunto de jugadores, de un conjunto de estrategias puras para cada jugador y de una función de pago. Por el otro lado, tenemos al modelo extensivo que incorpora más elementos que el rectangular para estudiar conflictos reales. El extensivo es un modelo menos estático que el rectangular, pues los jugadores tienen que tomar varias decisiones a lo largo del tiempo y, además, pueden aparecer en este modelo, problemas de falta de información que tengan algunos jugadores o azar, todo lo anterior dentro del mismo juego, es decir, que aporta más información que el modelo rectangular (Véase [A3]).

Como se establece desde el título, este texto aborda definiciones y algoritmos desde la perspectiva de los juegos rectangulares. El modelo extensivo no se atenderá aquí ya que implica otros planteamientos teóricos de los mismos temas, y agregar otros elementos a considerar en el modelo.

Definición 1.1.3.2. *Un juego rectangular es aquel juego* $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$ *donde :*

$$\varphi_j : \prod_{j=1}^{n} D_j \to R$$

Forma normal de un juego

La representación de un juego rectangular se le llama forma normal o estratégica, y varia según la cantidad de jugadores partícipes. Esto sucede como consecuencia de buscar la representación visual más cómoda para trabajar.

Todo juego n-personal en forma rectangular se representa con sus funciones de pagos, excepto en el caso de los juegos bipersonales, donde se representan con una matriz conocida como matriz de pagos. Cabe mencionar que los juegos rectangulares se nombran así como consecuencia natural de esta forma matricial (Véase [A3]).

Juegos en estrategias puras y en estrategias mixtas

Los juegos también pueden estudiarse dependiendo de las posibilidades que tienen los jugadores de elegir sus respuestas ante un juego, es decir, a la elección de una estrategia pura o mixta.

Un juego en estrategias puras es aquel en el que los jugadores solo tienen la posibilidad de elegir una sola estrategia por una única vez.

Un juego en estrategias mixtas se refiere a aquel donde los jugadores tienen la posibilidad o deber de combinar sus estrategias puras. Es decir, ahora el juego se repetirá muchas veces a lo largo del tiempo, dando un resultado conformado por muchas estrategias puras jugadas a lo largo del tiempo, o permitirá combinar estrategias por ocasión jugada, como por ejemplo, cuando se elige un portafolio de inversión y el capital se reparte en diferentes instrumentos financieros (Véase [A1] y [A5]).

1.2. Programación y software matemático

1.2.1. Programación y software matemático

La programación puede ser usada para realizar acciones lógicas y mecánicas a través de una computadora, como el manejo de máquinas, el desarrollo

de videojuegos, la automatización de tareas monótonas, etc. Y, aunque para todas las aplicaciones es ineludible el uso de matemáticas, el estudio de las matemáticas teóricas es una de las áreas de mayor impacto y aplicación para la programación.

Algunos lenguajes de programación incorporan elementos, usualmente agrupados en paquetes, que los hacen susceptibles para ser utilizados en la implementación computacional de algoritmos para muy diversas áreas matemáticas. No obstante, entre los lenguajes existen diferencias que afectan fundamentalmente a la cercanía notacional con respecto a las matemáticas, a la eficiencia con que ejecutan implementaciones o a las herramientas de desarrollo que acompañan el lenguaje.

Lo anterior ha hecho que no todos los lenguajes se hayan usado con la misma frecuencia en esta ciencia formal, e incluso existen diferencias de preferencia entre sus diversas ramas. Para cada propósito y condiciones que exija un problema serán más o menos adecuadas las características que ofrezca cada lenguaje de programación, sin embargo, las principales herramientas que hacen atractivos a ciertos lenguajes de programación para su uso matemático son la estabilidad, la eficiencia, el manejo de datos, la recursividad, y la legibilidad.

En la actualidad, algunos de los lenguajes de uso más frecuente en el quehacer matemático son:

- Matlab/Octave: Entre sus objetivos básicos se hallan la manipulación de matrices, la representación de datos y funciones, y la creación de interfaces de usuario (GUI).

- Mathematica/Maxima: Sistemas de álgebra computacional y de cálculo simbólico. Mathematica y Maxima, que es su versión libre, son también poderosos lenguajes de programación de propósito general. Cuentan con un amplio conjunto de funciones para hacer manipulación simbólica de polinomios, integración, derivación, expansión en series de potencias y de Fourier, entre otras funcionalidades.

- SPSS/PSPP: Programas estadísticos muy usados en las ciencias sociales y aplicadas, además de las empresas de investigación de mercado. El nombre originario de SPSS correspondía al acrónimo de Statistical Package for the Social Sciences.

- Python: Es un lenguaje de programación multiparadigma ampliamente usado en diferentes áreas científicas, cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

- R: Se trata de uno de los lenguajes de programación más utilizados en investigación por la comunidad estadística; siendo además muy popular en el campo de la minería de datos, la investigación biomédica, y las matemáticas financieras. Por todo lo anterior, R en la actualidad es uno de los lenguajes de programación más utilizados en investigación por la comunidad estadística. Como consecuencia directa, se ha convertido en la herramienta computacional por excelencia de ciertas comunidades, con el claro ejemplo de la comunidad actuarial mexicana y particularmente, la comunidad de actuarios egresados de la Universidad Nacional Autónoma de México.

1.2.2. Declaración de juegos en el entorno de R

Para explicar como se declara un juego en R ejemplificaremos con el siguiente conflicto de dos jugadores cuya argumentación y declaración del juego pueden encontrarse en 3.1.2 y D.2 respectivamente.

$$\begin{array}{cc} \text{J1}\backslash\text{J2} & \begin{array}{cc} V & NV \end{array} \\ \begin{array}{c} C \\ NC \end{array} & \begin{pmatrix} (-2,-2) & (4,4) \\ (6,-2) & (5,3) \end{pmatrix} \end{array}$$

El juego es bipersonal y tiene dos jugadores que, para este caso, se llaman "J1" y "J2". Por ello es que puede expresarse en esta forma matricial (Véase 1.1.3), de tal manera que en la esquina superior izquierda de la matriz se escribe el nombre identificador de ambos jugadores cuidando el orden, ya que este nos permitirá identificar que jugador se encuentra asociado con las filas y cual con las columnas. Para este caso "J1" se encuentra asociado a las filas y "J2" con las columnas.

Que "J1" se encuentre asociado a las filas significa que cada fila será una estrategia posible. En este caso, sus estrategias posibles con "C" y "NC". De manera análoga, "J2" tiene como estrategias posibles a "V" y "NV" que representan una columna cada una.

Como puede observarse, los elementos en la matriz son los resultados de la función de pagos evaluada en las combinaciones de las estrategias de los

jugadores, en este caso diremos que depende de la fila y la columna en cuestión. Es por esto que los elementos en la matriz contienen dos subelementos: el resultado del jugador asociado a las filas y el resultado del jugador asociado a las columnas, en ese orden.

El juego también puede expresarse en su forma estratégica (Véase 1.1.3), que consiste en listar los resultados de la función de pago para todas las combinaciones de las estrategias de los jugadores, es decir:

Se tienen 2 jugadores: J1 y J2 tales que,

$$\sigma_{J1} = \{C, NC\},\ \sigma_{J2} = \{V, NV\},$$

$$\begin{aligned}
\varphi(\ C, V)\ &= (\ -2, -2) \\
\varphi(\ C, NV)\ &= (\ 4, 4) \\
\varphi(\ NC, V)\ &= (\ 6, -2) \\
\varphi(\ NC, NV)\ &= (\ 5, 3)
\end{aligned}$$

Partiendo de la forma estratégica y previo al estudio con tecnologías de la información y cómputo de los temas de teoría de juegos, resulta indispensable proponer una forma estandarizada para declarar los juegos a usarse. A continuación se presenta dicha propuesta, que es resultado del análisis de diversas consideraciones, como la elección de "data frame" (Véase A.8) como la estructura de datos que permite la mejor manipulación de la información y que se ha usado en otras paqueterías y proyectos afines.

Se sugiere que para cualquier duda sobre programación se recurra a "A Algunas nociones de programación " *Apéndice A*, donde se incluyen materiales que con un estudio básico permiten entender los códigos en lo sucesivo.

Para empezar, se declara el número de jugadores en una variable, que para este ejemplo, y de manera no forzosa, es nombrada "j".

```
1 j <- 2
```

Se declaran los conjuntos de estrategias de cada jugador. Para cada uno se declara un vector que contiene sus estrategias declaradas como cadenas

de texto. Para este ejemplo se requieren dos vectores, y se nombran “pe_j1” y “pe_j2” como abreviatura de “perfil de estrategias del jugador 1” y “perfil de estrategias del jugador 2”.

```
pe_j1 <- c('V', 'NV')
pe_j2 <- c('C', 'NC')
```

Se calcula el número de filas que requerirá el data frame donde se guardará el juego multiplicando la longitud de los conjuntos de estrategias. Este valor se guarda en la variable llamada “filas”.

```
filas <- length(pe_j1)*length(pe_j2)
```

Se calcula el número de columnas que requerirá el data frame donde se guardará el juego multiplicando el número de jugadores (variable “j”) al doble. Este valor se guarda en la variable llamada “columnas”.

```
columnas <- j*2
```

Se inicializa un data frame que recibe de parámetro una matriz vacía que a su vez recibe como paramétros a la variable “filas” para el numero de filas (nrow), y la variable “columnas” para el numero de columnas (ncol). El data frame se nombra para el ejemplo “ej_juego”.

```
ej_juego <- data.frame(matrix(NA, nrow = filas, ncol = columnas
    ))
```

Se rotulan los encabezados de cada columna del data frame “ej_juego” . Es imperativo mencionar que esta forma de declarar un juego exige orden en la manera en que se declaran las columnas. Se deben declarar en dos bloques; el primero correspondiente a las estrategias para cada jugador y el segundo correspondiente a los pagos asociados. Por ello se nombran S_j’s y PHI’s, respectivamente.

```
names(ej_juego) <- c('S1', 'S2', 'PHI1', 'PHI2')
```

Muy parecido a como se escribiría el juego en su forma estratégica, declaramos las filas del data frame. Por ejemplo, para la primer fila se asigna un vector que contiene en orden: la estrategia del jugador 1, la estrategia del jugador 2, el resultado de la función de pago para el jugador 1 dadas las estrategias de los jugadores, y el resultado de la función de pago para el jugador 2 dadas las estrategias de los jugadores. Análogo para las demás filas.

```
ej_juego[1,] <- c('V', 'C', -2, -2)
ej_juego[2,] <- c('V', 'NC', 6, -2)
ej_juego[3,] <- c('NV', 'C', 4, 4)
ej_juego[4,] <- c('NV', 'NC', 5, 3)
```

Adicionalmente, si se desea declarar un juego en estrategias mixtas (Véase 1.1.3) se necesita el juego en estrategias puras, acompañado del perfil de estrategias inicial, que se declara de manera muy similar al juego. Por ejemplo:

Se calcula el número de filas que requerirá el data frame donde se guardará el perfil de estrategias sumando la longitud de los conjuntos de estrategias. Este valor se guarda en la variable llamada "filas".

```
filas <- length(pe_j1) + length(pe_j2)
```

Se inicializa un data frame que recibe de parámetro una matriz vacía que a su vez recibe como paramétros a la variable "filas" para el numero de filas (nrow), y el valor constante "2" para el numero de columnas (ncol). El data frame se nombra para el ejemplo "X_juego" debido a que X es la variable usada para expresar "perfil de estrategias".

```
X_juego <- data.frame(matrix(NA, nrow = filas, ncol = 2))
```

Se rotulan los encabezados de cada columna del data frame "X_juego" como "X" y "p".

```
names(X_juego) <- c('X', 'p')
```

Declaramos las filas del data frame. Por ejemplo, para la primer fila se asigna un vector que contiene en orden: la estrategia y el peso (probabilidad) de la estrategia. Análogo para las demás filas.

```
X_juego[1,] <- c('V', 1)
X_juego[2,] <- c('NV', 0)
X_juego[3,] <- c('C', 1)
X_juego[4,] <- c('NC', 0)
```

1.2.3. Visualización de los juegos en consola

Como se explicó en 1.1.3, la representación de un juego rectangular puede ser matricial o por estrategias. La representación adecuada se selecciona buscando que sea la más cómoda para trabajar el problema.

Por ello, se han preparado dos programas que permiten visualizar cómodamente cualquier juego que se haya declarado en R de la manera en que se

explicó en 1.2.2. Respectivamente, un programa devuelve la representación matricial y el otro la estratégica.

Estos programas se usarán a lo largo de los siguientes capítulos. Su código puede encontrarse en C.1 y, al igual que los juegos, pueden ser cargados en R de la manera que se explica en A.10.

Una vez cargado el juego , el perfil de estrategias inicial para los jugadores y el programa de visualizaciones, se puede obtener el juego con formato visual utilizando las funciones. A continuación se utiliza el mismo juego ejemplificador usado en 1.2.2 .

Entrada

Se declara la función "repMatr" con "ej_juego" como parámetro.

```
> repMatr(ej_juego)
```

Salida

Se obtiene una representación visual matricial del juego "ej_juego" .

```
      J1\\J2       V        NV
      C      (-2,-2)    (4,4)
      NC      (6,-2)    (5,3)
```

Entrada

Se declara la función "repEstr" con "ej_juego" como parámetro.

```
> repEstr(ej_juego)
```

Salida

Se obtiene una representación visual estratégica del juego "ej_juego" .

```
      Forma normal
      phi(C,V)   =  (-2,-2)
     phi(C,NV)   =    (4,4)
     phi(NC,V)   =    (6,-2)
    phi(NC,NV)   =    (5,3)
```

Entrada

Para visualizar un perfil de estrategias no es necesario utilizar una función. Basta con escribir el nombre de la variable tipo data frame donde se ha guardado. Para este ejemplo la variabla se llama "X_juego"

```
> X_juego
```

Salida

Se obtiene una representación visual del perfil de estrategias “X_juego” .

```
1                X p
2          cortes 1
3       descortes 0
4          Cortes 1
5       Descortes 0
```

Capítulo 2

Dominancias

2.0.1. Definición

Para un jugador en particular, existen casos en los cuales una estrategia puede resultar más beneficiosa que otra en todos los casos.

Dicho de otro modo, para todas las combinaciones de estrategias que hayan tomado los otros jugadores implicados en el juego, el jugador encontrará un mejor pago al elegir la primer estrategia en vez de la segunda (Véase [A5]) .

Partiendo de las definiciones iniciales (Véase 1.1.2.1 , 1.1.3.1 y 1.1.3.2) se define:

Definición 2.0.1.1. *Sean $\hat{\sigma}^j$ y $\bar{\sigma}^j$ son dos EP del jugador j, se dice que $\bar{\sigma}^j$ esta débilmente dominada por $\hat{\sigma}^j$ si:*

$$\varphi_j(\ \sigma|\hat{\sigma}^j)\ \geq \varphi_j(\ \sigma|\bar{\sigma}^j)\ ,\forall\sigma \in D$$

Adicionalmente, se dice que $\bar{\sigma}^j$ esta estrictamente dominada por $\hat{\sigma}^j$ si se cumple la desigualdad anterior de forma estricta:

$$\varphi_j(\ \sigma|\hat{\sigma}^j)\ > \varphi_j(\ \sigma|\bar{\sigma}^j)\ ,\forall\sigma \in D$$

2.0.2. Contexto de un problema

Juego de las falsas opciones para 3 jugadores

A diferencia de los ejemplos de las unidades anteriores, este juego no tiene

una argumentación práctica. Fue diseñado con fines ilustrativos.

El juego se basa en que para cada jugador, sus estrategias son combinaciones lineales únicamente multiplicadas por un escalar. Es decir que los pagos para las estrategias son x veces cualquier otra de sus estrategias.

El conflicto puede describirse de la siguiente forma, según las funciones de pago para todos los perfiles de estrategias en el juego:

Se tienen 3 jugadores cuyos nombres identificadores son: A, B y C tales que,

$$\sigma_A = \{a1, a2\},\ \sigma_B = \{b1, b2\},\ \sigma_C = \{c1, c2\}$$

$$\begin{aligned}
\varphi(\ a1, b1, c1) &= (\ 1, 4, -1)\\
\varphi(\ a1, b1, c2) &= (\ 5, 6, 0)\\
\varphi(\ a1, b2, c1) &= (\ 3, 2, 0)\\
\varphi(\ a1, b2, c2) &= (\ 7, 6, 1)\\
\varphi(\ a2, b1, c1) &= (\ 2, 8, 0)\\
\varphi(\ a2, b1, c2) &= (\ 10, 8, 2)\\
\varphi(\ a2, b2, c1) &= (\ 6, 4, -2)\\
\varphi(\ a2, b2, c2) &= (\ 14, 8, -1)
\end{aligned}$$

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.1.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_falsasOpciones.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_falsasOpciones.R")
```

Una vez cargado el juego, se puede visualizar en forma estratégica (Véase 1.2.3) de la siguiente manera.

Entrada

Se declara la función “repEstr” con “falsasOpciones” como parámetro.

```
> repEstr(falsasOpciones)
```

Salida

Se obtiene una representación visual estratégica del juego "falsasOpciones" que como se observa, es análogo a la descripción teórica del conflicto.

```
  Forma normal
phi(a1,b1,c1)  =   (1,4,-1)
phi(a1,b1,c2)  =    (5,6,0)
phi(a1,b2,c1)  =    (3,2,0)
phi(a1,b2,c2)  =    (7,6,1)
phi(a2,b1,c1)  =    (2,8,0)
phi(a2,b1,c2)  =   (10,8,2)
phi(a2,b2,c1)  =   (6,4,-2)
phi(a2,b2,c2)  =  (14,8,-1)
```

2.0.3. Solución del problema en forma teórica

Se observa que para los 3 jugadores, A, B y C, existen dos estrategias a tomar, donde una domina a la otra. Se procede a continuación a evidenciar estas dominancias para reducir de tamaño el juego y que pueda ser más fácil de manipular en otros análisis futuros. Partiendo de la teoría en 2.0.1 se tiene:

- Para el jugador A se observa que la estrategia *a2* domina fuertemente a la estrategia *a1*. Las estrategias del jugador A se ponen en negritas como apoyo visual para destacar que, en las desigualdades, son el elemento diferente del perfil de estrategias en cuestión.

$$\varphi_A(\ \mathbf{a1}, b1, c1) \ = 1 \ < \ 2 = \varphi_A(\ \mathbf{a2}, b1, c1)$$

$$\varphi_A(\ \mathbf{a1}, b1, c2) \ = 5 \ < \ 10 = \varphi_A(\ \mathbf{a2}, b1, c2)$$

$$\varphi_A(\ \mathbf{a1}, b2, c1) \ = 3 \ < \ 6 = \varphi_A(\ \mathbf{a2}, b2, c1)$$

$$\varphi_A(\ \mathbf{a1}, b2, c2) \ = 7 \ < \ 14 = \varphi_A(\ \mathbf{a2}, b2, c2)$$

$\therefore$ a1 está estrictamente dominada por a2

- Para el jugador B se observa que la estrategia *b1* domina débilmente a la estrategia *b2*.

$$\varphi_B(\ a1, \mathbf{b1}, c1) \ = 4 \ > \ 2 = \varphi_B(\ a1, \mathbf{b2}, c1)$$

$$\varphi_B(\ a1, \mathbf{b1}, c2) \ = 6 \ \geq \ 6 = \varphi_B(\ a1, \mathbf{b2}, c2)$$

$$\varphi_B(\ a2, \mathbf{b1}, c1) \ = 8 \ > \ 4 = \varphi_B(\ a2, \mathbf{b2}, c1)$$

$$\varphi_B(\ a2, \mathbf{b1}, c2) \ = 8 \ \geq \ 8 = \varphi_B(\ a2, \mathbf{b2}, c2)$$

$\therefore$ b2 está débilmente dominada por b1

- Para el jugador C se observa que la estrategia *c2* domina fuertemente a la estrategia *c1*.

$$\varphi_C(\ a1, b1, \mathbf{c1}) \ = -1 \ < \ 0 = \varphi_C(\ a1, b1, \mathbf{c2})$$

$$\varphi_C(\ a1, b2, \mathbf{c1}) \ = 0 \ < \ 1 = \varphi_C(\ a1, b2, \mathbf{c2})$$

$$\varphi_C(\ a2, b1, \mathbf{c1}) \ = 0 \ < \ 2 = \varphi_C(\ a2, b1, \mathbf{c2})$$

$$\varphi_C(\ a2, b2, \mathbf{c1}) \ = -2 \ < \ -1 = \varphi_C(\ a2, b2, \mathbf{c2})$$

$\therefore$ c1 está estrictamente dominada por c2

2.0.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.2. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/Dominancias.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/Dominancias.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue.

Entrada

Se declara la función “dominancias”, cuyos argumentos “juego” y “Visual” reciben, respectivamente, los argumentos “falsasOpciones” y “T” (Verdadero). Esto con el objetivo de que se reduzca el juego “falsasOpciones” por sus dominancias y se muestre el resultado con una guía “Visual” del proceso realizado.

```
> dominancias(falsasOpciones, visual = T)
```

Salida

Se obtiene la guía visual del proceso realizado en la reducción del juego “falsasOpciones” por sus dominancias y se muestra el resultado con una guía “Visual” del proceso realizado. Hasta el final se muestra, por supuesto, el juego reducido.

```
"PARA EL JUGADOR 1 TENEMOS:"
"Recopilando los perfiles de estrategias que conforman los
otros jugadores se tiene:"
  S2 S3 a1 a2
  b1 c1  1  2
  b1 c2  5 10
  b2 c1  3  6
  b2 c2  7 14

"Las dominancias para el jugador son:"
"a2 domina a a1"
"2 >= 1"  "10 >= 5" "6 >= 3"  "14 >= 7"
     [,1] [,2]
[1,]    0    0
[2,]    1    0

"Las estrategias no dominadas para el jugador son: "
"a2"

"El juego reducido para el jugador es: "
  S1 S2 S3 PHI1 PHI2 PHI3
  a2 b1 c1    2    8    0
  a2 b1 c2   10    8    2
  a2 b2 c1    6    4   -2
  a2 b2 c2   14    8   -1
```

```
26
27    "PARA EL JUGADOR  2  TENEMOS:"
28    "Recopilando los perfiles de estrategias que conforman los
      otros jugadores se tiene:"
29      S1 S3 b1 b2
30      a1 c1  4  2
31      a1 c2  6  6
32      a2 c1  8  4
33      a2 c2  8  8
34
35    "Las dominancias para el jugador son:"
36    "b1 domina a b2"
37    "4 >= 2" "6 >= 6" "8 >= 4" "8 >= 8"
38         [,1] [,2]
39    [1,]    0    1
40    [2,]    0    0
41
42    "Las estrategias no dominadas para el jugador son: "
43    "b1"
44
45    "El juego reducido para el jugador es: "
46      S1 S2 S3 PHI1 PHI2 PHI3
47      a1 b1 c1    1    4   -1
48      a1 b1 c2    5    6    0
49      a2 b1 c1    2    8    0
50      a2 b1 c2   10    8    2
51
52
53    "PARA EL JUGADOR  3  TENEMOS:"
54    "Recopilando los perfiles de estrategias que conforman los
      otros jugadores se tiene:"
55      S1 S2 c1 c2
56      a1 b1 -1  0
57      a1 b2  0  1
58      a2 b1  0  2
59      a2 b2 -2 -1
60
61    "Las dominancias para el jugador son:"
62    "c2 domina a c1"
63    "0 >= -1"  "1 >= 0"   "2 >= 0"   "-1 >= -2"
64         [,1] [,2]
65    [1,]    0    0
66    [2,]    1    0
67
68    "Las estrategias no dominadas para el jugador son: "
```

```
69      "c2"
70
71      "El juego reducido para el jugador es: "
72        S1 S2 S3 PHI1 PHI2 PHI3
73        a1 b1 c2    5    6    0
74        a1 b2 c2    7    6    1
75        a2 b1 c2   10    8    2
76        a2 b2 c2   14    8   -1
77
78
79
80      "EL JUEGO REDUCIDO PARA TODOS LOS JUGADORES ES:"
81        S1 S2 S3 PHI1 PHI2 PHI3
82        a2 b1 c2   10    8    2
83
```

Entrada

Así mismo se puede obtener únicamente el Data Frame del juego reducido por sus dominancias. Para ello en la entrada hay que establecer el argumento "visual" como "F" (Falso). El data frame resultante se asigna en la variable "FalsasOpcionesReducido" que después es llamada para mostrar su contenido en la salida.

```
1 > FalsasOpcionesReducido <- dominancias(falsasOpciones, visual =
      F)
2
3 > FalsasOpcionesReducido
```

Salida

Se obtiene el juego "falsasOpciones" reducido.

```
1         S1 S2 S3 PHI1 PHI2 PHI3
2         a2 b1 c2   10    8    2
```

También hay ocasiones en las que el juego resultante de la reducción de otro juego presenta nuevas dominancias, por lo cual se desea aplicar nuevamente el algoritmo de reducción de dominancias hasta llevar el juego original a su máxima reducción. Para eso se programó el algoritmo *reduccionPorDominancias* que se ejemplificará brevemente con el juego *empresasTec* que se encuentra en D.2 .

Entrada

Se declara la función "reduccionPorDominancias", cuyos argumentos "juego" y "Visual" reciben, respectivamente, los argumentos "empresasTec" y

"T" (Verdadero). Esto con el objetivo de que se reduzca el juego "empresasTec" lo máximo posible por sus dominancias y se muestre el resultado con una guía "Visual" del proceso realizado.

```
> reduccionPorDominancias(juego=empresasTec, visual=T)
```

Salida

Se obtiene la guía visual del proceso realizado en la reducción máxima del juego "empresasTec" por sus dominancias y se muestra el resultado con una guía "Visual" del proceso realizado. Hasta el final se muestra, por supuesto, el juego reducido.

```
"#### ITERACION 1 ####"
"*** PARA EL JUGADOR:  1"
"Recopilando los perfiles de estrategias que conforman los otros jugadores se tiene:"
            S2 Vetar No Vetar
    Competir     -2         4
 No Competir      6         5

"Las dominancias para el jugador son:"
     [,1] [,2]
[1,]    0    0
[2,]    0    0

"Las estrategias no dominadas para el jugador son: "
"Vetar"    "No Vetar"

"El juego reducido para el jugador es: "
        S1           S2 PHI1 PHI2
    Vetar     Competir   -2   -2
    Vetar No Competir    6   -2
 No Vetar     Competir    4    4
 No Vetar No Competir    5    3

"*** PARA EL JUGADOR:  2"
"Recopilando los perfiles de estrategias que conforman los otros jugadores se tiene:"
        S1 Competir No Competir
    Vetar        -2          -2
 No Vetar         4           3

"Las dominancias para el jugador son:"
"Competir domina a No Competir"
```

```
31      "-2 >= -2" "4 >= 3"
32           [,1] [,2]
33      [1,]    0    1
34      [2,]    0    0
35
36      "Las estrategias no dominadas para el jugador son: "
37      "Competir"
38
39      "El juego reducido para el jugador es: "
40             S1        S2 PHI1 PHI2
41         Vetar  Competir   -2   -2
42       No Vetar  Competir    4    4
43
44      "****EL JUEGO REDUCIDO PARA TODOS LOS JUGADORES ES:"
45             S1        S2 PHI1 PHI2
46         Vetar  Competir   -2   -2
47       No Vetar  Competir    4    4
48
49
50  "#### ITERACION 2 ####"
51      "*** PARA EL JUGADOR:  1"
52      "Recopilando los perfiles de estrategias que conforman los
        otros jugadores se tiene:"
53             S2 Vetar No Vetar
54       Competir    -2         4
55
56      "Las dominancias para el jugador son:"
57      "No Vetar domina a Vetar"
58      "4 >= -2"
59           [,1] [,2]
60      [1,]    0    0
61      [2,]    1    0
62
63      "Las estrategias no dominadas para el jugador son: "
64      "No Vetar"
65
66      "El juego reducido para el jugador es: "
67             S1        S2 PHI1 PHI2
68       No Vetar  Competir    4    4
69
70      "*** PARA EL JUGADOR:  2"
71      "Recopilando los perfiles de estrategias que conforman los
        otros jugadores se tiene:"
72             S1 Competir
73       No Vetar         4
```

```
74
75      "Las dominancias para el jugador son:"
76          [,1]
77      [1,]    0
78
79      "Las estrategias no dominadas para el jugador son: "
80      "Competir"
81
82      "El juego reducido para el jugador es: "
83             S1        S2 PHI1 PHI2
84       No Vetar Competir    4    4
85
86
87       "# No se puede reducir m'as el juego"
```

2.0.5. Otras consideraciones para ampliar

Los juegos que sirvieron de ejemplo para presentar las capacidades de los programas son singulares ya que muestran que reduciendo por dominancias un juego, se puede llegar a obtener la mejor respuesta de alguno o incluso de todos los jugadores. En el caso del último juego, se obtuvo como resultado un perfil de estrategias que corresponde la mejor respuesta conjunta de todos los jugadores, es decir, se obtuvo el equilibrio de Nash.

Es importante remarcar que la principal ventaja de las dominancias es poder reducir un juego para que sea más fácil seguir trabajando con él en análisis subsecuentes, con lo cuál los programas presentados adquieren valor, ya que permiten de forma casi instantánea reducir juegos tan complejos como se desee.

Por último, se sugiere usar ampliamente este algoritmo en otros juegos ya que el analizar y trabajar con un juego reducido resulta más cómodo y eficiente.

Capítulo 3

Equilibrio de Nash

3.1. Estrategias puras

3.1.1. Definición

Al resolver un juego pensando que los jugadores desean maximizar su ganancia, la estrategia donde se ubica su **mejor respuesta conjunta** se le conoce como equilibrio de Nash (Véase [A5]).

Para definir el equilibrio de Nash es entonces necesario definir **mejor respuesta**

Las nociones anterior se definen formalmente a continuación, partiendo de las definiciones iniciales (Véase 1.1.2.1 , 1.1.3.1 y 1.1.3.2).

Definición 3.1.1.1. *Dado $\hat{\sigma} \in D$, se dice que $\widetilde{\sigma}^j \in D_j$ es una mejor respuesta del jugador j a $\hat{\sigma}$ si*

$$\varphi_j(\ \hat{\sigma}|\widetilde{\sigma}^j)\ \geq \varphi_j(\ \hat{\sigma}|\sigma^j)\ , \forall \sigma_j \in D_j$$

Además se puede decir que $\widetilde{\sigma}^j \in D_j$ es mejor respuesta estricta del jugador a $\hat{\sigma}$ si, además de ser mejor respuesta:

$$\varphi_j(\ \hat{\sigma}|\widetilde{\sigma}^j)\ \geq \varphi_j(\ \hat{\sigma})$$

Definición 3.1.1.2. *σ^* en D es un equilibrio de Nash en estrategias puras (EP) si :*

$$\varphi_j(\ \sigma^*)\ \geq \varphi_j(\ \sigma^*|\sigma^j)\ \ , \forall \sigma^j \in D_j\ , \forall j \in N$$

Adicionalmente, se dice que σ^* en D es un equilibrio de Nash estricto en EP si :

$$\varphi_j(\ \sigma^*)\ > \varphi_j(\ \sigma^*|\sigma^j)\ \ , \forall \sigma^j \in D_j\ , \forall j \in N$$

Versando la definición anterior, un equilibrio de Nash es aquel en el cual si el jugador j juega con la estrategia σ_j^* tendrá un mejor o igual pago que si cambia de estrategia, suponiendo que los demás jugadores se mantengan jugando con el perfil de estrategias σ^*.

Al aplicar la definición de equilibrio de Nash en EP a un juego bipersonal expresado en su forma rectangular se desprende el siguiente algoritmo:

Es equilibrio de Nash todo aquel perfil de estrategias señalado por aquellas entradas de la matriz de pagos en la cual se cumpla simultáneamente que:

- Su primer componente (el pago para el jugador 1) sea el mayor valor de entre todos los primeros componentes de las demás entradas que se encuentren en su misma columna. Es decir, se busca la estrategia del jugador 1 (la fila) en la cual el pago es mayor dado que el jugador 2 elija cierta estrategia (la columna en cuestión).

- Su segundo componente (el pago para el jugador 2) sea el mayor valor de entre todos los segundos componentes de las demás entradas que se encuentren en su misma fila. Es decir, se busca la estrategia del jugador 2 (la columna) en la cual el pago es mayor dado que el jugador 1 elija cierta estrategia (la fila en cuestión).

3.1.2. Contexto de un problema

Empresas tecnológicas y política

Supóngase que existen dos empresas tecnológicas que son socias. La primera, que se identificara como "G", es estadounidense y provee software. La segunda, que se nombrará como "H", es originaria de China y produce hardware.

Su colaboración genera importantes ganancias para ambas. Por ejemplo, "G" y "H" el año pasado tuvieron utilidades derivadas únicamente de su colaboración de 6 y 3 mil millones de dólares, respectivamente.

Actualmente existe tensión internacional ya que el gobierno americano ha acusado a "H" de cometer espionaje y exige a "G" cortar lazos comerciales

y vetar a “H” de la distribución de su software.

La respuesta de “H” ha sido negar las acusaciones y, dada la difamación, revelar que ha estado desarrollando su propio software para competir con “G” a nivel mundial.

Ambas empresas deben tomar una decisión rápida. “G” tiene que decidir si vetará o no a “H”, por su parte, la empresa china debe decidir si entrará a competir en software o no.

Derivado de lo anterior, hay 4 posibles escenarios sobre las ganancias de ambas empresas a final de este año:

1. Si “G” veta a “H”, y “H” entra a competir en software: “G” tendría pérdidas estimadas de 2 mil millones debido a que las otras empresas chinas, que son la mayoría de sus clientes, respaldarían el proyecto de “H”. Por su parte “H” también tendría pérdidas estimadas de 2 mil millones ya que tendría que invertir mucho dinero en terminar el desarrollo e implementación de su software, sin percibir ganancias del mercado americano.

2. Si “G” veta a “H”, y “H” no entra a competir en software: “G” tendría las mismas ganancias del año anterior de 6 mil millones ya que a pesar de perder un importante contrato, sus socios lo respaldarían por acatar al gobierno. Por su parte “H” tendría pérdidas estimadas de 2 mil millones ya que al decidir a tiempo no competir, puede aceptar perder ese negocio y enfocarse en amortiguar el revés impulsando sus otros productos y servicios.

3. Si “G” no veta a “H”, y ”H” entra a competir en software: “G” tendría menos ganancias que el año anterior. estimadas en 4 mil millones ya que “H” le robaría algo de su mercado y tendría una sanción de mil millones por parte del gobierno estadounidense. Por su parte “H” crecería un poco respecto al año anterior, consiguiendo aproximadamente 4 mil millones ya que podría mantener sus productos, sus mercados y empezar a experimentar su propio software.

4. Si “G” no veta a “H”, y “H” no entra a competir en software: Se espera que ambos mantengan las mismas ganancias del año anterior, pero “G” enfrentaría una multa de mil millones por parte del gobierno

estadounidense. Por lo anterior, se espera que “G” gané 5 mil millones y “H” 3 mil millones.

El conflicto anterior es un juego que puede describirse con la siguiente matriz de pagos:

Ganancias del año pasado: 6 para ”G” y 3 para “H”.

$$\begin{array}{lcc} \text{G\H} & \textit{Competir} & \textit{No Competir} \\ \textit{Vetar} & (-2,-2) & (6,-2) \\ \textit{No Vetar} & (4,4) & (5,3) \end{array}$$

** Todas las unidades representan miles de millones de dolares.*

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.2.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_empresasTec.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_empresasTec.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función “repMatr” con “empresasTec” como parámetro.

```
> repMatr(empresasTec)
```

Salida

Se obtiene una representación visual matricial del juego “empresasTec” que como se observa, es análoga a la descripción teórica del conflicto.

```
   J1\\J2 Competir No Competir
   Vetar   (-2,-2)      (6,-2)
No Vetar     (4,4)       (5,3)
```

3.1.3. Solución del problema en forma teórica

Para hallar las mejores respuestas para las estrategias del jugador 1, que es "G".

$$\varphi_1(Vetar, Competir) = -2 \ < \ 4 = \varphi_1(No \ \ Vetar, Competir)$$

$\therefore \sigma_1 = No \ \ Vetar$ es la mejor respuesta del jugador 1 cuando $\sigma_2 = Competir$.

Ahora bien,

$$\varphi_1(Vetar, No \ \ Competir) = 6 \ > \ 5 = \varphi_1(No \ \ Vetar, No \ \ Competir)$$

$\therefore \sigma_1 = Vetar$ es la mejor respuesta del jugador 1 cuando $\sigma_2 = No \ \ Competir$.

Para hallar las mejores respuestas para las estrategias del jugador 2, que es "H".

$$\varphi_2(Vetar, Competir) = -2 \ = \ -2 = \varphi_2(Vetar, No \ \ Competir)$$

$\therefore \sigma_2 = Competir \ y \ No \ \ Competir$ son, ambas, mejor respuesta del jugador 2

$$\text{cuando } \sigma_1 = Vetar.$$

Ahora bien,

$$\varphi_2(No \ \ Vetar, Competir) = 4 \ > \ 3 = \varphi_2(No \ \ Vetar, No \ \ Competir)$$

$\therefore \sigma_2 = Competir$ es la mejor respuesta del jugador 2 cuando $\sigma_1 = No \ \ Vetar$.

En resumen, se tiene que los mejores perfiles de respuestas para el jugador 1 son (No vetar, Competir) y (Vetar, No Competir). Para el jugador 2 son (No vetar, Competir), (Vetar, Competir) y (Vetar, No Competir).

Destaca que la intersección de los conjuntos de mejores respuestas de los dos jugadores son los perfiles de estrategias: (No vetar, Competir) y (Vetar, No Competir). Esta observación es importante ya que, como se estableció anteriormente, el equilibrio de Nash es la mejor respuesta conjunta para los jugadores de un juego.

Partiendo de ello se comprueba a continuación que, por definición, ambos perfiles de estrategias son los Equilibrios de Nash del juego.

- Para $\sigma(No\ \ Vetar, Competir)$ se tiene que:

$$\varphi_1(No\ \ Vetar, Competir) = 4\ >\ \varphi_1(Vetar, Competir) = -2,$$
$$\varphi_2(No\ \ Vetar, Competir) = 4\ >\ \varphi_2(No\ \ Vetar, No\ \ Competir) = 3$$
$$\Rightarrow \sigma(No\ \ Vetar, Competir) \text{ es equilibrio de Nash.}$$

- Para $\sigma(Vetar, No\ \ Competir)$ se tiene que:

$$\varphi_1(Vetar, No\ \ Competir) = 6\ >\ \varphi_1(No\ \ Vetar, No\ \ Competir) = 5,$$
$$\varphi_2(Vetar, No\ \ Competir) = -2\ \geq\ \varphi_2(Vetar, Competir) = -2$$
$$\Rightarrow \sigma(Vetar, No\ \ Competir) \text{ es equilibrio de Nash.}$$

- Para $\sigma(Vetar, Competir)$ se tiene que:

$$\varphi_1(Vetar, Competir) = -2 < \varphi_1(No\ \ Vetar, Competir) = 4$$
$$\Rightarrow \sigma(Vetar, Competir) \text{ no es equilibrio de Nash.}$$

- Para $\sigma(No\ \ Vetar, No\ \ Competir)$ se tiene que:

$$\varphi_1(No\ \ Vetar, No\ \ Competir) = 5 < \varphi_1(Vetar, No\ \ Competir) = 6$$
$$\Rightarrow \sigma(No\ \ Vetar, No\ \ Competir) \text{ no es equilibrio de Nash.}$$

$$\therefore \sigma(No\ \ Vetar, Competir), \sigma(Vetar, No\ \ Competir)$$
$$\text{son los equilibrios de Nash del juego.}$$

3.1.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.3.1. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
1 "C:/MisDocumentos/Eq_Nash.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/Eq_Nash.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue. Recuérdese que se consideró a "G" como el jugador 1 y a "H" como el jugador 2.

Entrada

Para obtener las *mejores respuestas* del jugador 1 ante las estrategias del jugador 2, se usa la función *mejorRespuesta*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "1" para el parámetro "juego" y el parámetro "jugador", respectivamente.

```
> mejorRespuesta(empresasTec, 1)
```

Salida

Se obtiene un data frame con las *mejores respuestas* del jugador 1 ante las estrategias del jugador 2. Las columnas son las mismas del juego, y lo que nos dicen es que si el jugador 2 elige "No competir" la mejor respuesta del jugador 1 será "vetar". Por otro lado, si el jugador 2 elige "Competir" entonces la mejor respuesta del jugador 1 será "No vetar".

```
     S1        S2           PHI1  PHI2
     Vetar     No Competir  6     -2
     No Vetar  Competir     4     4
```

Entrada

Para obtener las *mejores respuestas* del jugador 2 ante las estrategias del jugador 1, se usa la función *mejorRespuesta*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "2" para el parámetro "juego" y el parámetro "jugador", respectivamente.

```
> mejorRespuesta(empresasTec, 2)
```

Salida

Se obtiene un data frame con las *mejores respuestas* del jugador 2 ante las estrategias del jugador 1. Se observa que si el jugador 1 elige "Vetar", la mejor respuesta del jugador 2 sera cualquiera de sus dos estrategias

posibles. Por otro lado, si el jugador 1 decide “No vetar”, entonces la mejor respuesta del jugador 1 será “Competir”.

```
1     S1          S2              PHI1   PHI2
2     Vetar       Competir        -2     -2
3     Vetar       No Competir     6      -2
4     No Vetar    Competir        4      4
```

Entrada

Para obtener el *equilibrio de Nash* del juego se usa la función *equilibrioNash*, con el argumento tipo data frame “empresasTec” para el parámetro “juego”.

```
1 > equilibrioNash(empresasTec)
```

Salida

Se obtiene un data frame con los *equilibrios de Nash* del juego. Se observa que el juego tiene dos equilibrios de Nash. Ambos formaron parte del conjunto de mejores respuestas de todos los jugadores del juego.

```
1     S1          S2              PHI1   PHI2
2     Vetar       No Competir     6      -2
3     No Vetar    Competir        4      4
```

3.1.5. Otros problemas para ampliar

El juego que sirvió de ejemplo para exhibir las capacidades de los programas es peculiar, ya que permite probar que la rutina es eficaz en juegos dónde hay múltiples *equilibrios de Nash*, es decir, encontrará tantos como existan en el juego. En particular se observa que hay dos equilibrios, donde uno es estricto y el otro no.

Por supuesto, se puede seguir experimentando con las capacidades de los programas, por lo cual, se sugiere que se prueben las funciones para otros juegos. Tanto aquellos en los que se tenga una intención o curiosidad singular, así como en los recopilados en “D.Recopilación de algunos juegos”.

Es de especial atención probar las rutinas con aquellos juegos en los que hay más de dos jugadores y aquellos en los que no exista equilibrio de Nash.

La idea fundamental de las estrategias mixtas es que el jugador no tiene porque elegir sólo una estrategia, sino una combinación de ellas asignándoles una proporción de su decisión a repartir. Es decir, repartirá un 100 por ciento de su decisión entre sus posibles estrategias, y, si se retoma la forma en que elegía en estrategias puras, es claro que ahí el 100 por ciento se destinaba únicamente a una decisión.

Gracias a este sistema de "pesos" en el perfil de estrategias del jugador, bien se puede expresar con una canasta de bienes a comprar en una decisión, o bien, como la probabilidad frecuentista de que una estrategia sea la mejor respuesta; con base a lo aprendido en la historia de un juego que se ha repetido numerosas veces en el pasado.

3.2. Mejor respuesta pura

El equilibrio de Nash en estrategias mixtas se ha estudiado intensamente y se han propuesto muchos algoritmos para hallarlo prácticamente desde que John Nash planteó una definición. Sin embargo, sin importar como se aborde el tema, el punto de partida es conocer la mejor respuesta que puede dar un jugador en cierto momento de la repetición de un juego. Por ello, a esa mejor respuesta en un tiempo determinado se le define como mejor respuesta pura(Véase [A5]).

La noción anterior se define formalmente a continuación, partiendo de las definiciones iniciales (Véase 1.1.2.1 , 1.1.3.1 y 1.1.3.2).

3.2.1. Definición

Definición 3.2.1.1. .

Dado el juego $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$ *finito, se dice que*

$$X^j \in R^{l_j}$$

es una estrategia mixta del jugador j si,

$$\forall \sigma^j \in D_j \text{ se cumple que } x^j_{\sigma^j} \geq 0 \ \wedge \ \sum_{\sigma^j \in D_j} x^j_{\sigma^j} = 1$$

donde

- l_j *denota el número de estrategias puras del jugador j,*
- $x^j_{\sigma^j}$ *se interpreta como el peso o probabilidad que el jugador j le asigna a su estrategia* σ^j .

Adicionalmente se establece a M_j *como el conjunto de estrategias mixtas del jugador j y M al producto cartesiano de los conjuntos* M_j . *A cada elemento de M se le llama un perfil de estrategias mixtas.*

Definición 3.2.1.2. .
Dado el juego $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$ *la función de pago esperado es la función definida como*

$$E[\ X^1, X^2, \ldots, X^n] = \sum_{\sigma^1,\sigma^2,\ldots,\sigma^n \in D} x^1_{\sigma^1} x^2_{\sigma^2} \ldots x^n_{\sigma^n} \varphi(\sigma^1, \sigma^2, \ldots, \sigma^n)$$

En particular, la función de pago esperado para algún jugador j, con el perfil de estrategias mixtas X se define como E_j. *Las funciones* E_j *corresponden a las funciones componente* φ_j *de E .*

Definición 3.2.1.3. .
Dado $X \in M, \hat{\sigma}$ *es una mejor respuesta pura del jugador j al perfil X, si*

$$\max_{\sigma^j \in D_j} E_j(X|\sigma_j) = E_j(X|\hat{\sigma_j})$$

Se denota como $R_j(X)$ *al conjunto de mejores respuestas puras del jugador j al perfil de estrategias mixtas X.*

3.2.2. Contexto de un problema

El conflicto siguiente se estudiará más adelante con detalle (Véase 3.3.2), por ahora solo es un ejercicio para ejemplificar.
El juego se describe con la siguiente matriz de pagos.

$$\begin{array}{cc} \text{J1}\backslash\text{J2} & \begin{array}{cc} c & d \end{array} \\ \begin{array}{c} C \\ D \end{array} & \begin{pmatrix} (-1,-1) & (0,2) \\ (2,0) & (-5,-5) \end{pmatrix} \end{array}$$

Se considera a $X = ((\frac{4}{7}, \frac{3}{7})(\frac{4}{7}, \frac{3}{7}))$

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.3.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_Vecinos.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_Vecinos.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "j_Vecinos" como parámetro.

```
> repMatr(j_Vecinos)
```

Salida

Se obtiene una representación visual matricial del juego "j_Vecinos" que como se observa, es análoga a la descripción teórica del conflicto.

```
    J1\\J2        c         d
    C       (-1,-1)    (0,2)
    D         (2,0)   (-5,-5)
```

Partiendo del perfil de estrategias "X" dado, se debe cargar (Véase A.10) el script en R con el código donde se declara dicho perfil de estrategias. Se puede usar cualquier perfil de estrategias que se desee, pero el considerado para este ejemplo se encuentra en **??**.

Entrada

Se carga el perfil de estrategias "X_mrp", posteriormente se escribe el nombre del objeto en la consola para obtener una visualización.

```
> source("C:/MisDocumentos/X_mrp.R")
> X_mrp
```

Salida

Se obtiene una representación visual del perfil de estrategias "X_mrp".

```
  X      p
c 0.5714
d 0.4286
C 0.5714
D 0.4286
```

3.2.3. Solución del problema en forma teórica

Se procede a calcular la mejor respuesta pura para cada jugador con el perfil de estrategias X.

- Para el jugador 1 se obtienen los pagos de sus respuestas puras:

$$E_1[(1,0)(4/7,3/7)] =$$

$$((1)(4/7)(-1)) + ((1)(3/7)(0)) + ((0)(4/7)(2)) + ((0)(3/7)(-5)) =$$

$$-\frac{4}{7} + 0 + 0 + 0 = -\frac{4}{7} = -0{,}5714$$

$$E_1[(0,1)(4/7,2/7)] =$$

$$((0)(4/7)(-1)) + ((0)(3/7)(0)) + ((1)(4/7)(2)) + ((1)(3/7)(-5)) =$$

$$0 + 0 + 8/7 - 15/7 = -\frac{7}{7}$$

Y como $-4/7 > -1$ entonces la mejor respuesta pura del jugador es el perfil (1,0), que corresponde a la estrategia *C*.

- Para el jugador 2 se obtienen los pagos de sus respuestas puras:

$$E_2[(4/7,3/7)(1,0)] =$$

$$((4/7)(1)(-1)) + ((3/7)(1)(0)) + ((4/7)(0)(2)) + ((3/7)(0)(-5)) =$$

$$-\frac{4}{7} + 0 + 0 + 0 = -\frac{4}{7} = -0{,}5714$$

$$E_2[(4/7,2/7)(0,1)] =$$

$$((4/7)(0)(-1)) + ((3/7)(0)(0)) + ((4/7)(1)(2)) + ((3/7)(1)(-5)) =$$

$$0 + 0 + 8/7 - 15/7 = -\frac{7}{7}$$

Y como $-4/7 > -1$ entonces la mejor respuesta pura del jugador es el perfil (1,0), que corresponde a la estrategia *c*.

3.2.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.3.2. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/EM_mrp.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/EM_mrp.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue.

Entrada

Para obtener las *respuestas puras* del jugador 1 se usa la función *respuestasPuras*, con el argumento tipo data frame "j_vecinos" y el argumento tipo entero "1" para los parámetros "juego" y "jugador" respectivamente.

```
> respuestasPuras(j_vecinos, 1)
```

Salida

Se obtienen las *respuestas puras* del jugador 1 en un data frame. La primer columna enlista las estrategias, y la segunda columna muestra el pago esperado asociado a la estrategia que está en la misma fila.

```
  X1      X2
C  C -0.5714
D  D      -1
```

Entrada

Para obtener la *mejor respuesta pura* del jugador 1 se usa la función *mejorRP*, dando como argumento el data frame resultante de la anterior instrucción. Esto se puede lograr guardando dicho data frame en una variable y pasandola como argumento, o bien, anidando funciones (Véase A. Algunas nociones de programación).

```
> mejorRP(respuestasPuras(ejemplo_EM, 1))
```

Salida

Se obtienen la cadena con la estrategia que es la *mejor respuesta pura* del jugador 1 para el juego.

```
"C"
```

Entrada

Para obtener las *respuestas puras* del jugador 2 se usa la función *respuestasPuras*, con el argumento tipo data frame "j_vecinos" y el argumento tipo entero "2" para los parámetros "juego" y "jugador" respectivamente.

```
> respuestasPuras(j_vecinos, 2)
```

Salida

Se obtienen las *respuestas puras* del jugador 2 en un data frame. La primer columna enlista las estrategias, y la segunda columna muestra el pago esperado asociado a la estrategia que está en la misma fila.

```
  X1      X2
C -0.5714
D    -1
```

Entrada

Para obtener la *mejor respuesta pura* del jugador 2 se usa la función *mejorRP*, dando como argumento el data frame resultante de la anterior instrucción. Esto se puede lograr guardando dicho data frame en una variable y pasándola como argumento, o bien, anidando funciones (Véase A. Algunas nociones de programación).

```
> mejorRP(respuestasPuras(ejemplo_EM, 2))
```

Salida

Se obtienen la cadena con la estrategia que es la *mejor respuesta pura* del jugador 2 para el juego.

```
"c"
```

3.2.5. Otros problemas para ampliar

La teoría de este apartado, así como el código que la acompaña, se utilizará como soporte de los siguientes algoritmos, por lo que la familiarización con el tópico es sumamente relevante.

3.3. Juego ficticio

Con éste algoritmo se busca estudiar las tendencias de comportamiento de los jugadores, expresando la experiencia que se tiene de dicho comportamiento, a través de ciertas estrategias mixtas, las estrategias mixtas obtenidas de la probabilidad clásica. Los jugadores reaccionan frente a tales estrategias eligiendo mejores respuestas puras. Por supuesto, si la tendencia de los jugadores converge hacia algún perfil de estrategias, se evidenciará que este es un equilibrio de Nash del juego (Véase [A5]).

A modo de formalización del concepto de equilibrio de Nash en estrategias puras, que será útil para este y siguientes temas, resulta enriquecedor enunciar su definición teórica.

Definición 3.3.0.1. *Un perfil de estrategias mixtas X^* es un equilibrio de Nash en estrategias mixtas (em) si*

$$E_j(X^*) \geq E_j(X^*|X^j), \forall X^* \in M_j, \forall j \in N$$

Podemos, entonces, decir que X^* es un equilibrio de Nash en estrategias mixtas (em) si se cumple que

$$\max_{X^j \in M_j} E_j(X^*|X^j) = E_j(X^*|X^j) = E_j(X^*), \forall j \in N$$

3.3.1. Algoritmo

Supóngase que un juego $(N, \{D_j\}_{\mathrm{j}\in\mathrm{N}}, \{\varphi_j\}_{\mathrm{j}\in\mathrm{N}})$ se repite durante los periodos t = 1, 2, ...

1. Cada jugador escoge una estrategia pura cualquiera, en el primer periodo. Si j escogió $\hat{\sigma^j}(1)$, se construye

   $$X(1) \in M, X(1) = (X^1(1), X^2(1), \ldots, X^n(1))$$

   donde $X^j(1) = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$, con el único 1 en la coordenada que corresponde a $\hat{\sigma^j}(1)$.

2. Si ningún jugador j tiene estrategias puras de mejor respuesta estricta a X (1), entonces el algoritmo termina con X(1).

3. Si algunos jugadores k tienen mejores respuestas estrictas a X (1), para t = 2, cada uno de ellos escoge una de dichas estrategias para el periodo 2, $\hat{\sigma^k}(2)$.

   Los jugadores k que no tienen mejores respuestas estrictas eligen $\hat{\sigma^k}(2) \in R_k(X(1))$, de tal manera que si $\hat{\sigma^k}(1) \in R_k(X(1))$, entonces $\hat{\sigma^k}(2) = \hat{\sigma^k}(1)$ (lo que se denomina hipótesis de inercia).

   Para cada jugador i, sea $n^i_{\hat{\sigma^i_l}}(2)$ el número de veces que el jugador i ha elegido σ^i_l en los periodos 1 y 2, y se define

$$X^i(2) = (\frac{n^i_{\hat{\sigma^i_1}}(2)}{2}, \frac{n^i_{\hat{\sigma^i_2}}(2)}{2}, \ldots, \frac{n^i_{\hat{\sigma^i_i}}(2)}{2})$$

   y como

$$X(2) = (X^1(2), X^2(2), \ldots, X^n(2))$$

4. Supóngase que, para t = m, se ha construido

$$X(m) = (X^1(m), X^2(m), \ldots, X^n(m))$$

   Si ningún jugador j tiene mejores respuestas estrictas a X(m), el algoritmo termina con X(m).

5. Si algunos jugadores k tienen mejores respuestas estrictas a X (m), para t = m+1, cada uno de ellos escoge una ellas como $\hat{\sigma^k}(m+1)$.

   Los jugadores k que no tienen mejores respuestas estrictas a X(m) eligen $\hat{\sigma^k}(m+1) \in R_k(X(m))$, de tal manera que si $\hat{\sigma^k}(m) \in R_k(X(1))$, entonces $\hat{\sigma^k}(m+1) = \hat{\sigma^k}(m)$ (nuevamente es la hipótesis de inercia).

   Para cada jugador i, sea $n^i_{\hat{\sigma^i_l}}(m+1)$ el número de veces que el jugador i ha elegido σ^i_l en los periodos t=1,2,...,m+1 y se define

$$X^i(m+1) = (\frac{n^i_{\hat{\sigma^i_1}}(m+1)}{m+1}, \frac{n^i_{\hat{\sigma^i_2}}(m+1)}{m+1}, \ldots, \frac{n^i_{\hat{\sigma^i_i}}(m+1)}{m+1})$$

   y como

$$X(m+1) = (X^1(m+1), X^2(m+1), \ldots, X^n(m+1))$$

6. Se repiten d, e y f para t = m+2

3.3.2. Contexto de un problema

Dos hombres que viven en el mismo edificio salen todos los días, de sus respectivos hogares, a la misma hora y los dos con una gran prisa, pues están a punto de llegar tarde a trabajar. Ambos tienen que sacar el coche del garaje común y no pueden hacerlo simultáneamente. Así, que cada día tienen que decidir si prefieren aguantar las ironías y reconvenciones de sus jefes, por llegar tarde a la oficina debido a sus cortesías con el vecino, o sufrir las maldiciones de dicho vecino si tratan de salir por delante, a como de lugar. Si los dos adoptan esa actitud descortés habrá una reyerta segura que significa mayor pérdida de tiempo.

El conflicto anterior se describe con la siguiente matriz de pagos:

$$\begin{array}{cc} \text{J1}\backslash\text{J2} & \begin{array}{cc} \textit{Cortes} & \textit{Descortes} \end{array} \\ \begin{array}{c} \textit{Cortes} \\ \textit{Descortes} \end{array} & \begin{pmatrix} (-1,-1) & (0,2) \\ (2,0) & (-5,-5) \end{pmatrix} \end{array}$$

** Todas las unidades representan el tiempo ganado o perdido.*

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.3.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_Vecinos.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_Vecinos.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "j_Vecinos" como parámetro.

```
> repMatr(j_Vecinos)
```

Salida

Se obtiene una representación visual matricial del juego "j_Vecinos" que como se observa, es análoga a la descripción teórica del conflicto.

```
    J1\\J2  cortes descortes
    Cortes (-1,-1)     (0,2)
 Descortes   (2,0)   (-5,-5)
```

Partiendo del perfil de estrategias "X" dado, se debe cargar (Véase A.10) el script en R con el código donde se declara dicho perfil de estrategias. Se puede usar cualquier perfil de estrategias que se desee, pero el considerado para este ejemplo se encuentra en el tercer apartado en D.3.

Entrada

Se carga el perfil de estrategias "X_jFict", posteriormente se escribe el nombre del objeto en la consola para obtener una visualización.

```
> source("C:/MisDocumentos/X_jFict.R")
> X_jFict
```

Salida

Se obtiene una representación visual del perfil de estrategias "X_jFict".

```
            X p
      cortes 1
   descortes 0
      Cortes 1
   Descortes 0
```

3.3.3. Solución del problema en forma teórica

- Para el primer periodo se elige $X(1)$ de forma arbitraria que, para este ejemplo, se considera $X(1) = ((1,0)(1,0))$. Ordenando lo anterior se tiene:

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$

Cuadro 3.1: Representación del juego ficticio

Teniendo esto, se procede a calcular la mejor respuesta pura para cada jugador.

- Para el jugador 1 se obtienen los pagos de sus respuestas puras:

$$E_1[(1,0)(1,0)] =$$

$$(1)(1)\varphi_1(\text{Cortés,Cortés}) + (1)(0)\varphi_1(\text{Cortés,Descortés}) +$$
$$(0)(1)\varphi_1(\text{Descortés,Cortés}) + (0)(0)\varphi_1(\text{Descortés,Descortés})$$
$$= ((1)(1)(-1)) + 0 + 0 + 0 = -1$$
$$E_1[(0,1)(1,0)] =$$
$$(0)(1)\varphi_1(\text{Cortés,Cortés}) + (0)(0)\varphi_1(\text{Cortés,Descortés}) +$$
$$(1)(1)\varphi_1(\text{Descortés,Cortés}) + (1)(0)\varphi_1(\text{Descortés,Descortés})$$
$$= 0 + 0 + ((1)(1)(2)) + 0 = 2$$

Y como $2 > -1$ entonces la mejor respuesta pura del jugador 1 es el perfil (0,1), que corresponde a la estrategia *Descortés*, y que usará para el siguiente periodo.

- Para el jugador 2 se obtienen los pagos de sus respuestas puras:

$$E_2[(1,0)(1,0)] =$$
$$(1)(1)\varphi_2(\text{Cortés,Cortés}) + (1)(0)\varphi_2(\text{Cortés,Descortés}) +$$
$$(0)(1)\varphi_2(\text{Descortés,Cortés}) + (0)(0)\varphi_2(\text{Descortés,Descortés})$$
$$= ((1)(1)(-1)) + 0 + 0 + 0 = -1$$
$$E_2[(1,0)(0,1)] =$$
$$(1)(0)\varphi_2(\text{Cortés,Cortés}) + (1)(1)\varphi_2(\text{Cortés,Descortés}) +$$
$$(0)(0)\varphi_2(\text{Descortés,Cortés}) + (0)(1)\varphi_2(\text{Descortés,Descortés})$$
$$= 0 + ((1)(1)(2)) + 0 + 0 = 2$$

Y como $2 > -1$ entonces la mejor respuesta pura del jugador 1 es el perfil (0,1), que corresponde a la estrategia *Descortés*, y que usará para el siguiente periodo.

- Para el segundo periodo, $X(2)$ proviene de las frecuencias relativas de las estrategias de cada jugador. Hasta este momento, cada jugador ha elegido cada estrategia una vez de las dos veces que ha tenido que elegir, por lo tanto, para ambos jugadores, $X(2) = ((\frac{1}{2}, \frac{1}{2})(\frac{1}{2}, \frac{1}{2}))$. Ordenando lo anterior se tiene:

  Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X.

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2},\frac{1}{2})$	Descortés	$(\frac{1}{2},\frac{1}{2})$

Cuadro 3.2: Representación del juego ficticio

- Para el jugador 1 se obtienen los pagos de sus respuestas puras:

$$E_1[(1,0)(1/2,1/2)] =$$

$$(1)(1/2)\varphi_1(\text{Cortés,Cortés}) + (1)(1/2)\varphi_1(\text{Cortés,Descortés}) +$$

$$(0)(1/2)\varphi_1(\text{Descortés,Cortés}) + (0)(1/2)\varphi_1(\text{Descortés,Descortés})$$

$$= ((1)(1/2)(-1)) + ((1)(1/2)(0)) + 0 + 0 = -\frac{1}{2}$$

$$E_1[(0,1)(1/2,1/2)] =$$

$$(0)(1/2)\varphi_1(\text{Cortés,Cortés}) + (0)(1/2)\varphi_1(\text{Cortés,Descortés}) +$$

$$(1)(1/2)\varphi_1(\text{Descortés,Cortés}) + (1)(1/2)\varphi_1(\text{Descortés,Descortés})$$

$$= 0 + 0 + ((1)(1/2)(2)) + ((1)(1/2)(-5)) = 2/2 - 5/2 = -\frac{3}{2}$$

Y como $-1/2 > -3/2$ entonces la mejor respuesta pura del jugador 1 es el perfil (1,0), que corresponde a la estrategia *Cortés*, y que usará para el siguiente periodo.

- Para el jugador 2 se obtienen los pagos de sus respuestas puras:

$$E_2[(1/2,1/2)(1,0)] =$$

$$(1/2)(1)\varphi_2(\text{Cortés,Cortés}) + (1/2)(0)\varphi_2(\text{Cortés,Descortés}) +$$

$$(1/2)(1)\varphi_2(\text{Descortés,Cortés}) + (1/2)(0)\varphi_2(\text{Descortés,Descortés})$$

$$= ((1/2)(1)(-1)) + 0 + ((1/2)(1)(0)) + 0 = -\frac{1}{2}$$

$$E_2[(1/2,1/2)(0,1)] =$$

$$(1/2)(0)\varphi_2(\text{Cortés,Cortés}) + (1/2)(1)\varphi_2(\text{Cortés,Descortés}) +$$

$$(1/2)(0)\varphi_2(\text{Descortés,Cortés}) + (1/2)(1)\varphi_2(\text{Descortés,Descortés})$$
$$= 0 + ((1/2)(1)(2)) + 0 + ((1/2)(1)(-5)) = 2/2 - 5/2 = -\frac{3}{2}$$

Y como $-1/2 > -3/2$ entonces la mejor respuesta pura del jugador 1 es el perfil (1,0), que corresponde a la estrategia *Cortés*, y que usará para el siguiente periodo.

- Para el tercer periodo, $X(3)$ proveniente de las frecuencias relativas de las estrategias de cada jugador es igual a $((\frac{2}{3}, \frac{1}{3})(\frac{2}{3}, \frac{1}{3}))$. Ordenando la información de los periodos hasta ahora se tiene:

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2}, \frac{1}{2})$	Descortés	$(\frac{1}{2}, \frac{1}{2})$
3	Cortés	$(\frac{2}{3}, \frac{1}{3})$	Cortés	$(\frac{2}{3}, \frac{1}{3})$

Cuadro 3.3: Representación del juego ficticio

Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X, como observación importante, se puede notar que ambos jugadores se han comportado y se seguirán comportando de la misma forma debido a que la matriz de pagos es simétrica y el juego es simétrico, a partir de ahora sólo se realizan las operaciones esenciales de un jugador (el jugador 1) que expresa a ambos.

- Para los jugadores se obtienen los pagos de sus respuestas puras:

$$E_j[(1,0)(2/3,1/3)] =$$
$$((1)(2/3)(-1)) + ((1)(1/3)(0)) + 0 + 0 = -\frac{2}{3}$$
$$E_j[(0,1)(2/3,1/3)] =$$
$$0 + 0 + ((1)(2/3)(2)) + ((1)(1/3)(-5)) = 4/3 - 5/3 = -\frac{1}{3}$$

Y como $-1/3 > -2/3$ entonces la mejor respuesta pura del jugador es el perfil (0,1), que corresponde a la estrategia *Descortés*, y que usará para el siguiente periodo.

- Para el cuarto periodo, $X(4)$ proveniente de las frecuencias relativas de las estrategias de cada jugador es igual a $((\frac{2}{4},\frac{2}{4})(\frac{2}{4},\frac{2}{4}))$. Ordenando la información de los periodos hasta ahora se tiene:

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2},\frac{1}{2})$	Descortés	$(\frac{1}{2},\frac{1}{2})$
3	Cortés	$(\frac{2}{3},\frac{1}{3})$	Cortés	$(\frac{2}{3},\frac{1}{3})$
4	Descortés	$(\frac{2}{4},\frac{2}{4})$	Descortés	$(\frac{2}{4},\frac{2}{4})$

Cuadro 3.4: Representación del juego ficticio

Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X,

- Para los jugadores se obtienen los pagos de sus respuestas puras:

$$E_j[(1,0)(2/4,2/4)] =$$

$$((1)(2/4)(-1)) + ((1)(2/4)(0)) + 0 + 0 = -\frac{2}{4}$$

$$E_j[(0,1)(2/4,2/4)] =$$

$$0 + 0 + ((1)(2/4)(2)) + ((1)(2/4)(-5)) = 4/4 - 10/4 = -\frac{6}{4}$$

Y como $-1/4 > -6/4$ entonces la mejor respuesta pura del jugador es el perfil (1,0), que corresponde a la estrategia *Cortés*, y que usará para el siguiente periodo.

- Para el quinto periodo, $X(5)$ proveniente de las frecuencias relativas de las estrategias de cada jugador es igual a $((\frac{3}{5},\frac{2}{5})(\frac{3}{5},\frac{2}{5}))$. Ordenando la información de los periodos hasta ahora se tiene:

Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X,

- Para los jugadores se obtienen los pagos de sus respuestas puras:

$$E_j[(1,0)(3/5,2/5)] =$$

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2},\frac{1}{2})$	Descortés	$(\frac{1}{2},\frac{1}{2})$
3	Cortés	$(\frac{2}{3},\frac{1}{3})$	Cortés	$(\frac{2}{3},\frac{1}{3})$
4	Descortés	$(\frac{2}{4},\frac{2}{4})$	Descortés	$(\frac{2}{4},\frac{2}{4})$
5	Cortés	$(\frac{3}{5},\frac{2}{5})$	Cortés	$(\frac{3}{5},\frac{2}{5})$

Cuadro 3.5: Representación del juego ficticio

$$((1)(3/5)(-1)) + ((1)(2/5)(0)) + 0 + 0 = -\frac{3}{5}$$

$$E_j[(0,1)(3/5, 2/5)] =$$

$$0 + 0 + ((1)(3/5)(2)) + ((1)(2/5)(-5)) = 6/5 - 10/5 = -\frac{4}{5}$$

Y como $-3/5 > -4/5$ entonces la mejor respuesta pura del jugador es el perfil (1,0), que corresponde a la estrategia *Cortés*, y que usará para el siguiente periodo.

- Para el sexto periodo, $X(6)$ proveniente de las frecuencias relativas de las estrategias de cada jugador es igual a $((\frac{4}{6},\frac{2}{6})(\frac{4}{6},\frac{2}{6}))$. Ordenando la información de los periodos hasta ahora se tiene:

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2},\frac{1}{2})$	Descortés	$(\frac{1}{2},\frac{1}{2})$
3	Cortés	$(\frac{2}{3},\frac{1}{3})$	Cortés	$(\frac{2}{3},\frac{1}{3})$
4	Descortés	$(\frac{2}{4},\frac{2}{4})$	Descortés	$(\frac{2}{4},\frac{2}{4})$
5	Cortés	$(\frac{3}{5},\frac{2}{5})$	Cortés	$(\frac{3}{5},\frac{2}{5})$
6	Cortés	$(\frac{4}{6},\frac{2}{6})$	Cortés	$(\frac{4}{6},\frac{2}{6})$

Cuadro 3.6: Representación del juego ficticio

Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X,

- Para los jugadores se obtienen los pagos de sus respuestas puras:

$$E_j[(1,0)(4/6,2/6)] =$$

$$((1)(4/6)(-1)) + ((1)(2/6)(0)) + 0 + 0 = -\frac{4}{6}$$

$$E_j[(0,1)(4/6,2/6)] =$$

$$0 + 0 + ((1)(4/6)(2)) + ((1)(2/6)(-5)) = 8/6 - 10/6 = -\frac{2}{6}$$

Y como $-4/6 < -2/6$ entonces la mejor respuesta pura del jugador es el perfil (0,1), que corresponde a la estrategia *Descortés*, y que usará para el siguiente periodo.

- Para el séptimo periodo, $X(7)$ proveniente de las frecuencias relativas de las estrategias de cada jugador es igual a $((\frac{4}{7},\frac{3}{7})(\frac{4}{7},\frac{3}{7}))$. Ordenando la información de los periodos hasta ahora se tiene:

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2},\frac{1}{2})$	Descortés	$(\frac{1}{2},\frac{1}{2})$
3	Cortés	$(\frac{2}{3},\frac{1}{3})$	Cortés	$(\frac{2}{3},\frac{1}{3})$
4	Descortés	$(\frac{2}{4},\frac{2}{4})$	Descortés	$(\frac{2}{4},\frac{2}{4})$
5	Cortés	$(\frac{3}{5},\frac{2}{5})$	Cortés	$(\frac{3}{5},\frac{2}{5})$
6	Cortés	$(\frac{4}{6},\frac{2}{6})$	Cortés	$(\frac{4}{6},\frac{2}{6})$
7	Descortés	$(\frac{4}{7},\frac{3}{7})$	Descortés	$(\frac{4}{7},\frac{3}{7})$

Cuadro 3.7: Representación del juego ficticio

Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X,

- Para los jugadores se obtienen los pagos de sus respuestas puras:

$$E_j[(1,0)(4/7,3/7)] =$$

$$((1)(4/7)(-1)) + ((1)(3/7)(0)) + 0 + 0 = -\frac{4}{7}$$

$$E_j[(0,1)(4/7,2/7)] =$$

$$0+0+((1)(4/7)(2))+((1)(3/7)(-5))=8/7-15/7=-\frac{7}{7}$$

Y como $-4/7>-1$ entonces la mejor respuesta pura del jugador es el perfil (1,0), que corresponde a la estrategia *Cortés*, y que usará para el siguiente periodo.

- Para el octavo periodo, $X(8)$ proveniente de las frecuencias relativas de las estrategias de cada jugador es igual a $((\frac{5}{8},\frac{3}{8})(\frac{5}{8},\frac{3}{8}))$. Ordenando la información de los periodos hasta ahora se tiene:

Periodo (t)	I	$X^I(t)$	II	$X^{II}(t)$
1	Cortés	$(1,0)$	Cortés	$(1,0)$
2	Descortés	$(\frac{1}{2},\frac{1}{2})$	Descortés	$(\frac{1}{2},\frac{1}{2})$
3	Cortés	$(\frac{2}{3},\frac{1}{3})$	Cortés	$(\frac{2}{3},\frac{1}{3})$
4	Descortés	$(\frac{2}{4},\frac{2}{4})$	Descortés	$(\frac{2}{4},\frac{2}{4})$
5	Cortés	$(\frac{3}{5},\frac{2}{5})$	Cortés	$(\frac{3}{5},\frac{2}{5})$
6	Cortés	$(\frac{4}{6},\frac{2}{6})$	Cortés	$(\frac{4}{6},\frac{2}{6})$
7	Descortés	$(\frac{4}{7},\frac{3}{7})$	Descortés	$(\frac{4}{7},\frac{3}{7})$
8	Cortés	$(\frac{5}{8},\frac{3}{8})$	Cortés	$(\frac{5}{8},\frac{3}{8})$

Cuadro 3.8: Representación del juego ficticio

Se procede a calcular la mejor respuesta pura para cada jugador con la nueva X,

- Para los jugadores se obtienen los pagos de sus respuestas puras:

$$E_j[(1,0)(5/8,3/8)]=$$

$$((1)(5/8)(-1))+((1)(3/8)(0))+0+0=-\frac{5}{8}$$

$$E_j[(0,1)(5/8,3/8)]=$$

$$0+0+((1)(5/8)(2))+((1)(3/8)(-5))=10/8-15/8=-\frac{5}{8}$$

Y como $-5/8=-5/8$ el algoritmo del juego ficticio termina aquí, con el perfil X(8)=$((\frac{5}{8},\frac{3}{8}),(\frac{5}{8},\frac{3}{8}))$, pues ininguno de los dos jugadores tiene mejores respuestas puras a X(8).

3.3.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.3.3. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/jFict.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/jFict.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

También es importante mencionar que para este script se ocupen algunas de las funcionalidades de los otros scripts que se ocupan anteriormente en este capítulo. Se deben cargar también.

Ahora bien, resolviendo ordenadamente el problema planteado. Recordando que el programa fue diseñado para retornar una lista de tres objetos, primero se guardará el resultado de nuestro ejemplo en una variable llamada çortesDescortes".

Entrada

Para obtener la simulación del *juego ficticio* del juego se usa la función *juegoFicticio*, con el argumento tipo data frame "j_jFict" y el argumento de tipo data frame "X_jFict" para los parámetros "juego" y "perfil de estrategias", respectivamente. El tercer parámetro es especial, específica que el algoritmo debe iterar un máximo de 180 veces.

```
> cortesDescortes <- juegoFicticio(ej_jFict, X_jFict, 180)
```

Salida

Se obtiene la tabla visual del juego ficticio, que es el primer objeto en la lista resultante.

```
> cortesDescortes[[1]]
 PeriodoT          I       X_I(t)          II      X_II(t)
        1     Cortes        (1,0)     cortes        (1,0)
        2 Descortes    (0.5,0.5) descortes    (0.5,0.5)
```

```
5          3     Cortes (0.667,0.333)     cortes (0.667,0.333)
6          4 Descortes     (0.5,0.5) descortes     (0.5,0.5)
7          5     Cortes     (0.6,0.4)     cortes     (0.6,0.4)
8          6     Cortes (0.667,0.333)     cortes (0.667,0.333)
9          7 Descortes (0.571,0.429) descortes (0.571,0.429)
10         8     Cortes (0.625,0.375)     cortes (0.625,0.375)
```

El segundo objeto en la lista resultante es el histórico de los perfiles de estrategia que adoptó el jugador 1.

```
1  > cortesDescortes[[2]]
2      t Cortes Descortes
3      1  1.000     0.000
4      2  0.500     0.500
5      3  0.667     0.333
6      4  0.500     0.500
7      5  0.600     0.400
8      6  0.667     0.333
9      7  0.571     0.429
10     8  0.625     0.375
```

El tercer y último objeto en la lista resultante es el histórico de los perfiles de estrategia que adoptó el jugador 2.

```
1  > cortesDescortes[[3]]
2      t Cortes Descortes
3      1  1.000     0.000
4      2  0.500     0.500
5      3  0.667     0.333
6      4  0.500     0.500
7      5  0.600     0.400
8      6  0.667     0.333
9      7  0.571     0.429
10     8  0.625     0.375
```

3.3.5. Otros problemas para ampliar

Se invita a comprobar el algoritmo para ejemplos más variados, empezando por un juego que no sea simétrico, y modificando el perfil de estrategias inicial de cada jugador. También se sugiere aprovechar los históricos de los perfiles de estrategias que entrega el programa para graficarlos y tener una mejor referencia visual.

No sobra mencionar que este algoritmo no es bueno desde el punto de vista numérico, pues no converge fácilmente, sin embargo debido a la forma tan

adecuada en que simula una forma muy razonable de actuar de los jugadores no sólo ha inspirado nuevos algoritmos más sofisticados, sino que ha jugado un papel muy interesante en el estudio de patrones de conducta sociales, a pesar de lo laborioso que resulta al momento de usar la teoría para resolver un problema.

3.4. Función de reajuste de Nash

Los jugadores que se están enfrentando repetidamente en un conflicto usando estrategias mixtas pueden preferir cambiar más lentamente dichas estrategias, en lugar de abandonar bruscamente los pesos (probabilidades) que estaban utilizando como sucede en el *juego ficticio*, en aras de dar una mejor respuesta, sea pura o mixta (Véase [A5]).

Es decir, pueden conformarse con elegir nuevas estrategias mixtas que les permitan mejorar, quizá no lo más posible, su esperanza de pago, pero evitando caer en ciclos o procesos que parecieran no tener una tendencia clara.

La función construida por Nash tiene la característica de reajustar los pesos asignados a cada estrategia pura σ_j de cualquier jugador j, de acuerdo a que tan "buena" demostró ser, en el sentido de la cantidad en que σ_j hubiera incrementado el pago esperado, en el caso de que j la hubiera usado, respecto al perfil de estrategias mixtas que se estaba usando por todos los jugadores.

3.4.1. Algoritmo

Puntualizando lo anterior, los jugadores de un juego rectangular

$$(N, \{D_j\}_{\mathrm{j}\in\mathrm{N}}, \{\varphi_j\}_{\mathrm{j}\in\mathrm{N}})$$

pueden llevar a cabo un proceso análogo al de juego ficticio, pero en lugar de ir corrigiendo sus estrategias mixtas del periodo anterior con una mejor respuesta, reajustarlas con la función T de Nash que se describe en breve. Y buscando cual es el resultado final de este nuevo algoritmo, si es que dicho algoritmo termina.

Es claro que X^* es un perfil de terminación si y solo si X^* es un punto fijo de T , es decir si $T(X^*) = X^*$.

Precisando, la función T de Nash es $T : M \longrightarrow M$, definida como

$$T(X) = \hat{X},$$

donde sus componentes son

$$\hat{x}^j_{\bar{\sigma}_j} = \frac{x^j_{\bar{\sigma}_j} + c^j_{\bar{\sigma}_j}(X)}{1 + \sum_{\sigma\in D_j} c^j_{\sigma_j}(X)} \text{ , con}$$

$$c^j_{\sigma_j}(X) = \text{máx}\{E_j(X|\sigma_j) - E_j(X), 0\}$$

3.4.2. Contexto de un problema

Para esta sección se emplea un juego sin un contexto práctico, seleccionado únicamente con fines ilustrativos.

El conflicto se describe con la siguiente matriz de pagos:

$$\begin{array}{cc} \text{J1}\backslash\text{J2} & \begin{array}{cc} c & d \end{array} \\ \begin{array}{c} A \\ B \end{array} & \begin{pmatrix} (2,4) & (3,-3) \\ (4,1) & (5,2) \end{pmatrix} \end{array}$$

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.4.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_reajusteNash.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_reajusteNash.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "j_reajusteNash" como parámetro.

```
> repMatr(j_reajusteNash)
```

Salida

Se obtiene una representación visual matricial del juego "j_reajusteNash" que como se observa, es análoga a la descripción teórica del conflicto.

```
      J1\\J2       c         d
           A  (2,4)  (3,-3)
           B  (4,1)   (5,2)
```

Partiendo del perfil de estrategias "X" dado, se debe cargar (Véase A.10) el script en R con el código donde se declara dicho perfil de estrategias. Se puede usar cualquier perfil de estrategias que se desee, pero el considerado para este ejemplo se encuentra en el segundo apartado en D.4.

Entrada

Se carga el perfil de estrategias "X_reajusteNash", posteriormente se escribe el nombre del objeto en la consola para obtener una visualización.

```
> source("C:/MisDocumentos/X_reajusteNash.R")
> X_reajusteNash
```

Salida

Se obtiene una representación visual del perfil de estrategias "X_reajusteNash".

```
      X p
      A 1
      B 0
      c 1
      d 0
```

3.4.3. Solución del problema en forma teórica

Para este problema, por la complejidad que implica, se procederá sólo durante 3 iteraciones.

Iteración 1 Empezamos con $X_0 = [(1,0)(1,0)]$

- Se obtiene $E_j(X_0)$ para el jugador 1.

$$E_1[(1,0)(1,0)] =$$

$$(1)(1)(2) + (1)(0)(3) + (0)(1)(4) + (0)(0)(5)$$

$$2 + 0 + 0 + 0 = 2$$

- Se obtiene $c^j_{\sigma_j}(X_0)$ para la estrategia A del jugador 1.

$$c^1_{\sigma_1=A}(X_0) = \text{máx}\{E_1[(1,0)(1,0)] - E_1(X_0), 0\}$$

$$c^1_{\sigma_1=A}(X_0) = \text{máx}\{E_1(X_0) - E_1(X_0), 0\}$$

$$c^1_{\sigma_1=A}(X_0) = \text{máx}\{0, 0\} = 0$$

- Se obtiene $c^j_{\sigma_j}(X_0)$ para la estrategia B del jugador 1.

$$c^1_{\sigma_1=B}(X_0) = \text{máx}\{E_1[(0,1)(1,0)] - E_1(X_0), 0\}$$

Donde

$$E_1[(0,1)(1,0)] = (0)(1)(2) + (0)(0)(3) + (1)(1)(4) + (1)(0)(5) = 4$$

Por lo que

$$c^1_{\sigma_1=B}(X_0) = \text{máx}\{4-2, 0\} = 2$$

- Se obtiene $E_j(X_0)$ para el jugador 2.

$$E_2[(1,0)(1,0)] =$$

$$(1)(1)(4) + (1)(0)(-3) + (0)(1)(1) + (0)(0)(2)$$

$$4 + 0 + 0 + 0 = 2$$

- Se obtiene $c^j_{\sigma_j}(X_0)$ para la estrategia c del jugador 2.

$$c^2_{\sigma_2=c}(X_0) = \text{máx}\{E_2[(1,0)(1,0)] - E_2(X_0), 0\}$$

$$c^2_{\sigma_2=c}(X_0) = \text{máx}\{E_2(X_0) - E_2(X_0), 0\} = 0$$

- Se obtiene $c^j_{\sigma_j}(X_0)$ para la estrategia d del jugador 2.

$$c^2_{\sigma_2=d}(X_0) = \text{máx}\{E_2[(1,0)(0,1)] - E_2(X_0), 0\}$$

Donde

$$E_2[(1,0)(0,1)] = (1)(0)(4) + (1)(1)(-3) + (0)(0)(1) + (0)(1)(2) = -3$$

Por lo que

$$c^2_{\sigma_2=d}(X_0) = \text{máx}\{-3-2, 0\} = 0$$

- Se obtiene $\sum_{\sigma \in D_j} c^j_{\sigma_j}(X_0)$ para el jugador 1.

$$\sum_{\sigma \in D_1} c^1_{\sigma_1}(X_0) = c^1_{\sigma_1=A}(X_0) + c^1_{\sigma_1=B}(X_0) = 0 + 2 = 2$$

- Se obtiene $\sum_{\sigma \in D_j} c^j_{\sigma_j}(X_0)$ para el jugador 2.

$$\sum_{\sigma \in D_2} c^2_{\sigma_2}(X_0) = c^2_{\sigma_2=c}(X_0) + c^1_{\sigma_2=d}(X_0) = 0 + 0 = 0$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia A del jugador 1.

$$\hat{x}^1_{\bar{\sigma}_1=A} = \frac{x^1_{\bar{\sigma}_1=A} + c^1_{\bar{\sigma}_1=A}(X_0)}{1 + \sum_{\sigma \in D_1} c^1_{\sigma_1}(X_0)} = \frac{1+0}{1+2} = 1/3 \approx 0.333$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia B del jugador 1.

$$\hat{x}^1_{\bar{\sigma}_1=B} = \frac{x^1_{\bar{\sigma}_1=B} + c^1_{\bar{\sigma}_1=B}(X_0)}{1 + \sum_{\sigma \in D_1} c^1_{\sigma_1}(X_0)} = \frac{0+2}{1+2} = 2/3 \approx 0.666$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia c del jugador 2.

$$\hat{x}^2_{\bar{\sigma}_2=c} = \frac{x^2_{\bar{\sigma}_2=c} + c^1_{\bar{\sigma}_2=c}(X_0)}{1 + \sum_{\sigma \in D_2} c^1_{\sigma_2}(X_0)} = \frac{1+0}{1+0} = 1$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia d del jugador 2.

$$\hat{x}^2_{\bar{\sigma}_2=d} = \frac{x^2_{\bar{\sigma}_2=d} + c^1_{\bar{\sigma}_2=d}(X_0)}{1 + \sum_{\sigma \in D_2} c^1_{\sigma_2}(X_0)} = \frac{0+0}{1+0} = 0$$

$$\therefore X_1 = [(1/3, 2/3)(1, 0)]$$

Y como $X_0 \neq X_1$ entonces no se llegó a un punto fijo, por lo que se continúa con la siguiente iteración.

Iteración 2 Empezamos con $X_1 = [(1/3,2/3)(1,0)]$

- Se obtiene $E_j(X_1)$ para el jugador 1.

$$E_1[(1/3, 2/3)(1, 0)] =$$

$$(1/3)(1)(2) + (1/3)(0)(3) + (2/3)(1)(4) + (2/3)(0)(5)$$

$$2/3 + 0 + 8/3 + 0 = 10/3$$

- Se obtiene $c^j_{\sigma_j}(X_1)$ para la estrategia A del jugador 1.

$$c^1_{\sigma_1=A}(X_1) = \text{máx}\{E_1[(1, 0)(1, 0)] - E_1(X_1), 0\}$$

$$c^1_{\sigma_1=A}(X_1) = \text{máx}\{2 - 10/3, 0\} = 0$$

- Se obtiene $c^j_{\sigma_j}(X_1)$ para la estrategia B del jugador 1.

$$c^1_{\sigma_1=B}(X_1) = \text{máx}\{E_1[(0,1)(1,0)] - E_1(X_1), 0\}$$

Donde

$$E_1[(0,1)(1,0)] = (0)(1)(2) + (0)(0)(3) + (1)(1)(4) + (1)(0)(5) = 4$$

Por lo que

$$c^1_{\sigma_1=B}(X_0) = \text{máx}\{4 - (10/3), 0\} = 2/3$$

- Se obtiene $E_j(X_1)$ para el jugador 2.

$$E_2[(1/3, 2/3)(1,0)] =$$

$$(1/3)(1)(4) + (1/3)(0)(-3) + (2/3)(1)(1) + (2/3)(0)(2)$$

$$4/3 + 0 + 2/3 + 0 = 2$$

- Se obtiene $c^j_{\sigma_j}(X_1)$ para la estrategia c del jugador 2.

$$c^2_{\sigma_2=c}(X_1) = \text{máx}\{E_2[(1/3, 2/3)(1,0)] - E_2(X_1), 0\}$$

$$c^2_{\sigma_2=c}(X_1) = \text{máx}\{E_2(X_1) - E_2(X_1), 0\} = 0$$

- Se obtiene $c^j_{\sigma_j}(X_1)$ para la estrategia d del jugador 2.

$$c^2_{\sigma_2=d}(X_1) = \text{máx}\{E_2[(1/3, 2/3)(0,1)] - E_2(X_1), 0\}$$

Donde

$$E_2[(1/3, 2/3)(0,1)] =$$

$(1/3)(0)(4)+(1/3)(1)(-3)+(2/3)(0)(1)+(2/3)(1)(2) = -3/3+4/3 = 1/3$

Por lo que

$$c^2_{\sigma_2=d}(X_1) = \text{máx}\{1/3 - 2, 0\} = 0$$

- Se obtiene $\sum_{\sigma \in D_j} c^j_{\sigma_j}(X_1)$ para el jugador 1.

$$\sum_{\sigma \in D_1} c^1_{\sigma_1}(X_1) = c^1_{\sigma_1=A}(X_1) + c^1_{\sigma_1=B}(X_1) = 0 + 2/3 = 2/3$$

- Se obtiene $\sum_{\sigma \in D_j} c^j_{\sigma_j}(X_1)$ para el jugador 2.

$$\sum_{\sigma \in D_2} c^2_{\sigma_2}(X_1) = c^2_{\sigma_2=c}(X_1) + c^1_{\sigma_2=d}(X_1) = 0 + 0 = 0$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia A del jugador 1.

$$\hat{x}^1_{\bar{\sigma}_1=A} = \frac{x^1_{\bar{\sigma}_1=A} + c^1_{\bar{\sigma}_1=A}(X_1)}{1 + \sum_{\sigma \in D_1} c^1_{\sigma_1}(X_1)} = \frac{1/3 + 0}{1 + 2/3} = 1/5 = 0.2$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia B del jugador 1.

$$\hat{x}^1_{\bar{\sigma}_1=B} = \frac{x^1_{\bar{\sigma}_1=B} + c^1_{\bar{\sigma}_1=B}(X_1)}{1 + \sum_{\sigma \in D_1} c^1_{\sigma_1}(X_1)} = \frac{2/3 + 2/3}{1 + 2/3} = 4/5 = 0.8$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia c del jugador 2.

$$\hat{x}^2_{\bar{\sigma}_2=c} = \frac{x^2_{\bar{\sigma}_2=c} + c^1_{\bar{\sigma}_2=c}(X_1)}{1 + \sum_{\sigma \in D_2} c^1_{\sigma_2}(X_1)} = \frac{1 + 0}{1 + 0} = 1$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia d del jugador 2.

$$\hat{x}^2_{\bar{\sigma}_2=d} = \frac{x^2_{\bar{\sigma}_2=d} + c^1_{\bar{\sigma}_2=d}(X_1)}{1 + \sum_{\sigma \in D_2} c^1_{\sigma_2}(X_1)} = \frac{0 + 0}{1 + 0} = 0$$

$$\therefore X_2 = [(1/5, 4/5)(1, 0)]$$

Y como $X_1 \neq X_2$ entonces no se llegó a un punto fijo, por lo que se continúa con la siguiente iteración.

Iteración 3 Empezamos con $X_2 = [(1/5,4/5)(1,0)]$

- Se obtiene $E_j(X_2)$ para el jugador 1.

$$E_1[(1/5, 4/5)(1, 0)] =$$

$$(1/5)(1)(2) + (1/5)(0)(3) + (4/5)(1)(4) + (4/5)(0)(5)$$

$$2/5 + 0 + 16/5 + 0 = 18/5$$

- Se obtiene $c^j_{\sigma_j}(X_2)$ para la estrategia A del jugador 1.

$$c^1_{\sigma_1=A}(X_2) = \text{máx}\{E_1[(1,0)(1,0)] - E_1(X_2), 0\}$$

$$c^1_{\sigma_1=A}(X_2) = \text{máx}\{2 - 18/5, 0\} = 0$$

- Se obtiene $c^j_{\sigma_j}(X_2)$ para la estrategia B del jugador 1.

$$c^1_{\sigma_1=B}(X_2) = \text{máx}\{E_1[(0,1)(1,0)] - E_1(X_2), 0\}$$

Donde

$$E_1[(0,1)(1,0)] = (0)(1)(2) + (0)(0)(3) + (1)(1)(4) + (1)(0)(5) = 4$$

Por lo que

$$c^1_{\sigma_1=B}(X_0) = \text{máx}\{4 - (18/5), 0\} = 2/5$$

- Se obtiene $E_j(X_2)$ para el jugador 2.

$$E_2[(1/5, 4/5)(1,0)] =$$

$$(1/5)(1)(4) + (1/5)(0)(-3) + (4/5)(1)(1) + (4/5)(0)(2)$$

$$4/5 + 0 + 4/5 + 0 = 8/5$$

- Se obtiene $c^j_{\sigma_j}(X_2)$ para la estrategia c del jugador 2.

$$c^2_{\sigma_2=c}(X_2) = \text{máx}\{E_2[(1/5, 4/5)(1,0)] - E_2(X_2), 0\}$$

$$c^2_{\sigma_2=c}(X_2) = \text{máx}\{E_2(X_2) - E_2(X_2), 0\} = 0$$

- Se obtiene $c^j_{\sigma_j}(X_2)$ para la estrategia d del jugador 2.

$$c^2_{\sigma_2=d}(X_2) = \text{máx}\{E_2[(1/5, 4/5)(0,1)] - E_2(X_2), 0\}$$

Donde

$$E_2[(1/5, 4/5)(0,1)] =$$

$$(1/5)(0)(4)+(1/5)(1)(-3)+(4/5)(0)(1)+(4/5)(1)(2) = -3/5+8/5 = 1$$

Por lo que

$$c^2_{\sigma_2=d}(X_2) = \text{máx}\{1 - 2, 0\} = 0$$

- Se obtiene $\sum_{\sigma\in D_j} c^j_{\sigma_j}(X_2)$ para el jugador 1.

$$\sum_{\sigma\in D_1} c^1_{\sigma_1}(X_2) = c^1_{\sigma_1=A}(X_2) + c^1_{\sigma_1=B}(X_2) = 0 + 2/5 = 2/5$$

- Se obtiene $\sum_{\sigma\in D_j} c^j_{\sigma_j}(X_2)$ para el jugador 2.

$$\sum_{\sigma\in D_2} c^2_{\sigma_2}(X_2) = c^2_{\sigma_2=c}(X_2) + c^1_{\sigma_2=d}(X_2) = 0 + 0 = 0$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia A del jugador 1.

$$\hat{x}^1_{\bar{\sigma}_1=A} = \frac{x^1_{\bar{\sigma}_1=A} + c^1_{\bar{\sigma}_1=A}(X_2)}{1 + \sum_{\sigma\in D_1} c^1_{\sigma_1}(X_2)} = \frac{1/5 + 0}{1 + 2/5} = 1/5 \approx 0.1429$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia B del jugador 1.

$$\hat{x}^1_{\bar{\sigma}_1=B} = \frac{x^1_{\bar{\sigma}_1=B} + c^1_{\bar{\sigma}_1=B}(X_2)}{1 + \sum_{\sigma\in D_1} c^1_{\sigma_1}(X_2)} = \frac{4/5 + 2/5}{1 + 2/5} = 6/7 \approx 0.8571$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia c del jugador 2.

$$\hat{x}^2_{\bar{\sigma}_2=c} = \frac{x^2_{\bar{\sigma}_2=c} + c^1_{\bar{\sigma}_2=c}(X_2)}{1 + \sum_{\sigma\in D_2} c^1_{\sigma_2}(X_2)} = \frac{1 + 0}{1 + 0} = 1$$

- Se obtiene $\hat{x}^j_{\bar{\sigma}_j}$ para la estrategia d del jugador 2.

$$\hat{x}^2_{\bar{\sigma}_2=d} = \frac{x^2_{\bar{\sigma}_2=d} + c^1_{\bar{\sigma}_2=d}(X_2)}{1 + \sum_{\sigma\in D_2} c^1_{\sigma_2}(X_2)} = \frac{0 + 0}{1 + 0} = 0$$

$$\therefore X_3 = [(1/7, 6/7)(1, 0)]$$

Y como $X_2 \neq X_3$ entonces no se llegó a un punto fijo, por lo que se debe continuar con las siguientes iteraciones necesarias ...

3.4.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.3.4. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/reajusteNash.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/reajusteNash.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

También es importante mencionar que para este script se ocupan algunas de las funcionalidades de los otros scripts que se ocupan anteriormente en este capítulo. Se deben cargar también.

Ahora bien, resolviendo el problema planteado:
Entrada

Recordando que el programa fue diseñado para retornar la lista de estrategias mixtas reajustadas para cada tiempo, especificamos que se deberá iterar un máximo de 20 veces si no se encuentra un punto fijo antes. Se usa la función *reajusteNash*, con el argumento tipo data frame "ej_reajusteNash" y el argumento de tipo data frame "X_reajusteNash" para los parámetros "juego" y "perfil de estrategias", respectivamente. El tercer parámetro es especial, específica que el algoritmo debe iterar un máximo de 20 veces.

```
> reajusteNash(ej_reajusteNash, X_reajusteNash, 20)
```

Salida

Se obtiene la tabla visual del juego con Reajuste Nash. Cada columna cuyo nombre empieza con t, representa los pesos de las estrategias en ese tiempo. Como se puede observar, este juego no cae en ningún ciclo, ni en un estado absorbente, pero se observa una convegencia gradual hacia las estrategias "B" y "d", para los jugadores 1 y 2 respectivamente,

```
X p         t1  t2         t3         t4
A 1 0.3333333 0.2 0.1428571 0.1111111
B 0 0.6666667 0.8 0.8571429 0.8888889
c 1 1.0000000 1.0 1.0000000 1.0000000
d 0 0.0000000 0.0 0.0000000 0.0000000
X p          t5          t6          t7          t8
A 1 0.09090909 0.07692308 0.06666667 0.05882353
B 0 0.90909091 0.92307692 0.93333333 0.94117647
c 1 0.90000000 0.72262774 0.56546573 0.44740318
d 0 0.10000000 0.27737226 0.43453427 0.55259682
X p          t9         t10         t11         t12
A 1 0.05263158 0.04761905 0.04347826 0.0400000
B 0 0.94736842 0.95238095 0.95652174 0.9600000
c 1 0.36172485 0.29908961 0.25236420 0.2166988
d 0 0.63827515 0.70091039 0.74763580 0.7833012
X p         t13         t14         t15         t16
A 1 0.03703704 0.03448276 0.03225806 0.03030303
B 0 0.96296296 0.96551724 0.96774194 0.96969697
c 1 0.18886807 0.16671099 0.14875324 0.13396784
d 0 0.81113193 0.83328901 0.85124676 0.86603216
X p         t17         t18         t19         t20
A 1 0.02857143 0.02702703 0.02564103 0.02439024
B 0 0.97142857 0.97297297 0.97435897 0.97560976
c 1 0.12162411 0.11119163 0.10227808 0.09458825
d 0 0.87837589 0.88880837 0.89772192 0.90541175
```

3.4.5. Otros problemas para ampliar

Como se puede observar, el ejemplo presentado no logra tener un final en los primeros 20 tiempos pero muestra una clara tenencia convergente hacia el equilibrio de Nash en estrategias puras del juego.

Se invita a comprobar el algoritmo para ejemplos más variados y modificando el perfil de estrategias inicial de cada jugador. También se sugiere aprovechar los históricos de los perfiles de estrategias que entrega el programa para graficarlos y tener una mejor referencia visual para identificar una convergencia o un comportamiento distinto.

No sobra mencionar que este algoritmo, al igual que su antecesor, el juego ficticio, no es el mejor desde el punto de vista numérico, pues no converge fácilmente.

Parecería una pérdida de tiempo esta función de reajuste de Nash, sin em-

bargo, resulta que es la clave para la demostración del teorema fundamental de los juegos no cooperativos finitos, el *teorema de Nash*, que establece que siempre existe al menos un equilibrio de Nash en estrategias mixtas.

La importancia de este programa radica en la facilidad que proporciona para poder obtener un resultado que si se resolviera de forma manual sería demasiado laborioso y entorpecería la obtención de conclusiones en el estudio de algún juego, por lo cual, se invita a probarlo para facilitar la solución de juegos que resulten muy trabajosos.

Capítulo 4

Estrategias conservadoras y máximo asegurable

4.1. Estrategias puras

4.1.1. Definición

Para un jugador en particular, se dice que "x" es el valor *asegurable* de una de sus estrategias cuando, sin importar qué hagan los demás jugadores, su ganancia con la estrategia elegida es al menos "x". Dicho de otra forma, el asegurable de una estrategia, es lo peor que le puede suceder al jugador en turno si elige dicha estrategia (Véase [A5]).

Ahora bien, el jugador tiene varias estrategias disponibles para elegir, por lo que que para cada estrategia tendrá un asegurable. Si se agrupan todos esos asegurables se tendrá el *conjunto de valores asegurables* para el jugador.

Consecuencia de lo anterior, en el conjunto habrá un asegurable que sea mayor o igual a todos los demás. Ese valor es el *máximo asegurable*.

Finalmente, hay que recordar que el máximo asegurable del jugador es el asegurable de una o varias estrategias del jugador. Esa o esas estrategias se les denomina *estrategia conservadora*.

Las nociones anteriores se definen formalmente a continuación, Partiendo de las definiciones iniciales (Veáse 1.1.2.1 , 1.1.3.1 y 1.1.3.2).

Definición 4.1.1.1. *Se dice que x es asegurable en EP para j si*

$$\exists \hat{\sigma}^j \in D_j : \varphi_j(\ \sigma|\hat{\sigma}^j)\ \geq x, \forall \sigma \in D, x \in R$$

Definición 4.1.1.2. *Adicionalmente, se define al conjunto de los valores asegurables para j, correspondientes a todos los* $\sigma^j \in D_j$ *, como*

$$A'_j = \{x \in R | x \text{ es asegurable en EP para } j\}$$

Definición 4.1.1.3. A'_j *es un subconjunto de los reales no vacío y acotado superiormente, por lo que tiene un supremo que se denomina máximo asegurable en EP, se expresa como* v'_j *y se define como:*

$$v'_j = \max_{\sigma^j \in D_j} \min_{\sigma \in D} \varphi_j(\ \sigma | \sigma^j)$$

Definición 4.1.1.4. *Finalmente* $\hat{\sigma}^j$ *es una estrategia conservadora en EP para el jugador j, si:*

$$\varphi_j(\ \sigma | \hat{\sigma}^j)\ \geq v'_j, \hat{\sigma}^j \in D_j, \forall \sigma \in D$$

Existen estrategias conservadoras para cualquier juego y para cada jugador participante.

4.1.2. Contexto de un problema

Para esta sección se vuelve a emplear el ejemplo *Empresas tecnológicas y política* que se planteó en 3.1.2.

Retomando, el conflicto se describe con la siguiente matriz de pagos:
Se tienen 2 jugadores cuyos nombres identificadores son "G" y "H" tales que,

$$\begin{array}{lc} \text{G\H} & \begin{array}{cc} \textit{Competir} & \textit{No Competir} \end{array} \\ \begin{array}{l} \textit{Vetar} \\ \textit{No Vetar} \end{array} & \left(\begin{array}{cc} (-2,-2) & (6,-2) \\ (4,4) & (5,3) \end{array} \right) \end{array}$$

(Ganancias del año pasado: 6 para "G" y 3 para "H")
** Todas las unidades representan miles de millones de dolares.*

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.2.
Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_empresasTec.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_empresasTec.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "empresasTec" como parámetro.

```
> repMatr(empresasTec)
```

Salida

Se obtiene una representación visual matricial del juego "empresasTec" que como se observa, es análoga a la descripción teórica del conflicto.

```
   J1\\J2 Competir No Competir
   Vetar   (-2,-2)      (6,-2)
No Vetar     (4,4)       (5,3)
```

4.1.3. Solución del problema en forma teórica

Primero se encuentran los asegurables para el jugador 1, que es "G":

$$\varphi_1(\ \sigma|\hat{\sigma}^1 = \text{“Vetar”})\ = \{-2, 6\} \geq -2, \forall\sigma \in D$$

$\therefore$ -2 es el asegurable para la estrategia "Vetar".

$$\varphi_1(\ \sigma|\hat{\sigma}^1 = \text{“No Vetar”})\ = \{4, 5\} \geq 4, \forall\sigma \in D$$

$\therefore$ 4 es el asegurable para la estrategia "No Vetar".

Como consecuencia inmediata, se tiene el conjunto de valores asegurables para el jugador 1, "G", y el máximo asegurable.

$$A_1' = \{-2, 4\}$$

$$\therefore v_1' = max(\ A_1')\ = 4$$

Finalmente, la estrategia conservadora se comprueba por la definición:

$$\varphi_1(\ \sigma|\hat{\sigma}^1 = \text{“No Vetar”})\ = \{4, 5\} \geq 4 = v_1', \forall\sigma \in D$$

$\therefore \hat{\sigma}^1 =$ "No Vetar" es la estrategia conservadora de "G" para el juego dado.

Siguiendo un procedimiento análogo para el jugador 2 "H", primero se encuentran los asegurables para el jugador 2, que es "H".

$$\varphi_2(\ \sigma|\hat{\sigma}^1 = \text{“Competir”})\ = \{-2, 4\} \geq -2, \forall \sigma \in D$$

$\therefore$ -2 es el asegurable para la estrategia "Competir".

$$\varphi_2(\ \sigma|\hat{\sigma}^1 = \text{“No Competir”})\ = \{-2, 3\} \geq -2, \forall \sigma \in D$$

$\therefore$ -2 es el asegurable para la estrategia "No Competir".

Como consecuencia inmediata, se tiene el conjunto de valores asegurables para el jugador 2, "H", y el máximo asegurable.

$$A_2' = \{-2, -2\}$$

$$\therefore v_2' = max(\ A_2')\ = -2$$

Finalmente, se tienen dos estrategias conservadoras, las cuales se comprueban por la definición:

$$\varphi_2(\ \sigma|\hat{\sigma}^2 = \text{“Competir”})\ = \{-2, 4\} \geq -2 = v_2', \forall \sigma \in D$$

$$\varphi_2(\ \sigma|\hat{\sigma}^2 = \text{“No Competir”})\ = \{-2, 3\} \geq -2 = v_2', \forall \sigma \in D$$

$\therefore \hat{\sigma}_1^2 =$ "Competir" $\wedge\ \hat{\sigma}_2^2 =$ "No Competir" son estrategias conservadoras de "G"

4.1.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.4.1. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/ECyMA.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/ECyMA.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue. Hay que recordar que se consideró a "G" como el jugador 1 y a "H" como el jugador 2.

Entrada

Para obtener el *asegurable* del jugador 1 si elige la estrategia "Vetar" en el juego de las empresas tecnológicas se usa la función *asegurable*, con el argumento tipo data frame "empresasTec", el argumento tipo entero "1" y el argumento tipo cadena "Vetar" para los parámetros "juego", "numero de jugador" y "estrategia del jugador", respectivamente.

```
> asegurable(empresasTec,1,"Vetar")
```

Salida

Se obtiene el valor *asegurable* del jugador 1 al elegir la estrategia "Vetar" en el juego de las empresas tecnológicas.

```
    -2
```

Entrada

Para obtener el *asegurable* del jugador 1 si elige la estrategia "No Vetar" en el juego de las empresas tecnológicas se usa la función *asegurable*, con el argumento tipo data frame "empresasTec", el argumento tipo entero "1" y el argumento tipo cadena "No Vetar" para los parámetros "juego", "numero de jugador" y "estrategia del jugador", respectivamente.

```
> asegurable(empresasTec,1,"No Vetar")
```

Salida

Se obtiene el valor *asegurable* del jugador 1 al elegir la estrategia "No Vetar" en el juego de las empresas tecnológicas.

```
    -4
```

Entrada

Para obtener el *asegurable* del jugador 2 si elige la estrategia "Competir" en el juego de las empresas tecnológicas se usa la función *asegurable*, con el argumento tipo data frame "empresasTec", el argumento tipo entero "1" y el argumento tipo cadena "No Vetar" para los parámetros "juego", "numero de jugador" y "estrategia del jugador", respectivamente.

```
> asegurable(empresasTec,2,"Competir")
```

Salida

Se obtiene el valor *asegurable* del jugador 1 al elegir la estrategia "No Vetar" en el juego de las empresas tecnológicas.

```
-2
```

Entrada

Para obtener el *asegurable* del jugador 2 si elige la estrategia "No Competir" en el juego de las empresas tecnológicas se usa la función *asegurable*, con el argumento tipo data frame "empresasTec", el argumento tipo entero "1" y el argumento tipo cadena "No Vetar" para los parámetros "juego", "numero de jugador" y "estrategia del jugador", respectivamente.

```
> asegurable(empresasTec,2,"No Competir")
```

Salida

Se obtiene el valor *asegurable* del jugador 1 al elegir la estrategia "No Vetar" en el juego de las empresas tecnológicas.

```
-2
```

Entrada

Para obtener el *conjunto de asegurables* del jugador 1 en el juego de las empresas tecnológicas se usa la función *conjAseg_j*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "1" para los parámetros "juego" y "numero de jugador".

```
> conjAseg_j(empresasTec,1)
```

Salida

Se obtiene el data frame del *conjunto de asegurables* del jugador 1 en el juego de las empresas tecnológicas. La primer columna representa los valores asegurables de cada estrategia enlistada en la segunda columna.

```
  conj    estr
-2      Vetar
 4    No Vetar
```

Entrada

Para obtener el *conjunto de asegurables* del jugador 2 en el juego de las empresas tecnológicas se usa la función *conjAseg_j*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "2" para los parámetros "juego" y "numero de jugador".

```
> conjAseg\_j(empresasTec,2)
```

Salida

Se obtiene el data frame del *conjunto de asegurables* del jugador 2 en el juego de las empresas tecnológicas.

```
    conj       estr
    -2       Competir
    -2    No Competir
```

Entrada

Para obtener el *máximo asegurable* del jugador 1 en el juego de las empresas tecnológicas se usa la función *maxAseg*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "1" para los parámetros "juego" y "numero de jugador".

```
> maxAseg(empresasTec,1)
```

Salida

Se obtiene el valor del *máximo asegurable* del jugador 1 en el juego de las empresas tecnológicas.

```
    4
```

Entrada

Para obtener el *máximo asegurable* del jugador 2 en el juego de las empresas tecnológicas se usa la función *maxAseg*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "2" para los parámetros "juego" y "numero de jugador".

```
> maxAseg(empresasTec,2)
```

Salida

Se obtiene el valor del *máximo asegurable* del jugador 2 en el juego de las empresas tecnológicas.

```
    -2
```

Entrada

Para obtener la *estrategia conservadora* del jugador 1 en el juego de las empresas tecnológicas se usa la función *estraConserv*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "1" para los parámetros "juego" y "numero de jugador".

```
> estraConserv(empresasTec,1)
```

Salida

Se obtiene la o las *estrategias conservadoras* del jugador 1 en el juego de las empresas tecnológicas.

```
    "No Vetar"
```

Entrada

Para obtener la *estrategia conservadora* del jugador 2 en el juego de las empresas tecnológicas se usa la función *estraConserv*, con el argumento tipo data frame "empresasTec" y el argumento tipo entero "2" para los parámetros "juego" y "numero de jugador".

```
> estraConserv(empresasTec,2)
```

Salida

Se obtiene la o las *estrategias conservadoras* del jugador 2 en el juego de las empresas tecnológicas.

```
    "Competir"      "No Competir"
```

4.1.5. Otros problemas para ampliar

El juego que sirvió de ejemplo para presentar las capacidades de los programas permite probar que la rutina es eficaz en juegos dónde hay múltiples *estrategias asegurables* para algún jugador, es decir, encontrará tantas *estrategias asegurables* como existan en el juego. En particular para este juego se observa que para el segundo jugador sus dos estrategias posibles resultan ser *estrategias conservadoras*.

Adicionalmente, es de especial atención probar las rutinas con aquellos juegos en los que hay más de dos jugadores, como por ejemplo, el "Disparejo", entre otros.

4.2. Método algebraico de obtención de estrategias conservadoras en juegos 2x2

Un jugador también puede comportarse en forma conservadora en estrategias mixtas. Es decir, tiene la opción de considerar, cuando mezcla estrategias en el corto o en en el largo plazo, que todos los demás jugadores están unificados en su contra, como si se tratara de un juego bipersonal de suma cero. Las estrategias conservadoras mixtas del jugador j son aquellas que les permiten ganar lo máximo posible, cuando se basan sólo en sus propias fuerzas, sin esperar apoyo de nadie más. A dicha máxima ganancia se le llama el máximo asegurable y es un marco de referencia que le permitirá decidir si es conveniente abandonar su "actuar solitario" para aceptar un pacto propuesto por otros jugadores (Véase [A5] y [A2]).

Al enfrentar, ahora, a conjuntos infinitos de estrategias, ya no resulta inmediato asegurar la existencia y validez de los conceptos teóricos, resultados y demostraciones mencionadas para el mismo tema en la sección de estrategias puras, sino que se requiere una matemática más compleja, que queda fuera de los fines y alcances de este material.

El enfoque del máximo asegurable, como ya se ha establecido, está basado en que cada jugador j construye un juego imaginario, bipersonal de suma cero; es un juego en el que j se concibe sólo contra el mundo, es decir, en ese juego, él mismo es el primer jugador y el segundo son todos los jugadores restantes unidos contra él, buscando lograr que j gane lo menos posible. Por ello, los métodos para calcular el máximo asegurable de un jugador son aquellos que resuelven los juegos bipersonales de suma cero. En aras de no complicar la notación, al exponer estos métodos de solución, aparecerán sólo dos jugadores, pero del segundo, excepto en el caso de que, en realidad, se esté frente a un juego bipersonal de suma cero, no nos interesan sus estrategias conservadoras y su máximo asegurable, pues es un jugador imaginario que únicamente existe en la mente del jugador j.

A modo de formalización de los conceptos anteriores, y previo al planteamiento del método para hallar las estrategias conservadoras en juegos bipersonales de suma cero, es enriquecedor enunciar su origen teórico, el teorema de Von Neumann.

Teorema de Von Neumann
Todo juego bipersonal de suma cero es antagónico en estrategias mixtas. O,

de forma equivalente,en un juego bipersonal de suma cero $(\{1,2\},\{D_1,D_2\},\varphi)$, $(\varphi_2=-\varphi_1)$, *se cumple que:*

$$\max_{X^1\in M_1}(\min_{X^2\in M_2}E(X^1,X^2))=\min_{X^2\in M_2}(\max_{X^1\in M_1}E(X^1,X^2))$$

4.2.1. Desarrollo del método

Supóngase que el "juego bipersonal de suma cero imaginario" $A^{(j)}$ que construye el jugador j tiene como matriz de pagos (Véase [A5]):

$$\begin{pmatrix}(a_{11},-a_{11}) & (a_{12},-a_{12})\\ (a_{21},-a_{21}) & (a_{22},-a_{22})\end{pmatrix}$$

Si $A^{(j)}$ tiene un punto silla en (i,k), i es una estrategia pura conservadora de j y a_{ik} es su máximo asegurable.

Supóngase ahora el caso en que la matriz no tiene punto silla.

Si j, el jugador del que se calculará alguna estrategia conservadora y el máximo asegurable, elige la estrategia mixta (x, 1-x) y el jugador imaginario elige (y, 1-y), el pago para j será:

$$E_j((x,1-x),(y,1-y))=$$
$$xya_{11}+x(1-y)a_{12}+(1-x)ya_{21}+(1-x)(1-y)a_{22}=$$
$$y(xa_{11}+(1-x)a_{21})+(1-y)(xa_{12}+(1-x)a_{22})$$

Cuando ambos "jugadores" han elegido una pareja de estrategias conservadoras, a las que se denota como $(x^*,1-x^*)$ y $(y^*,1-y^*)$, respectivamente. Dado que el juego es simétrico y antagónico, se tiene que,

Si X^* *en* M *es un perfil de estrategias puras (em) entonces* $E_j(X^*)=v_j$

y por lo tanto:

$$y^*(x^*a_{11}+(1-x^*)a_{21})+(1-y^*)(x^*a_{12}+(1-x^*)a_{22})=v_j$$

Pero, además,

$$x^*a_{11}+(1-x^*)a_{21}\geq v_j \text{ y}$$

$$x^* a_{12} + (1 - x^*) a_{22} \geq v_j$$

Ninguna de las dos desigualdades puede ser estricta pues no podría cumplirse la igualdad anterior, así que se tienen las ecuaciones:

$$x^* a_{11} + (1 - x^*) a_{21} = v_j \text{ y}$$

$$x^* a_{12} + (1 - x^*) a_{22} = v_j$$

Al resolver el sistema de ecuaciones, se obtienen las soluciones de x^* y v_j:

$$x^* = \frac{a_{22} - a_{21}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

Se debe puntualizar un escenario usual; si el resultado algebraico del calculo del perfil de estrategia conservadora fuera mayor a 1, se le asignará el máximo peso posible, es decir, 1. Si, por otro lado, el resultado es un número negativo, el peso de la estrategia sería 0, el mínimo posible. Se debe recordar que las estrategias tienen pesos acotados en [0,1].

$$v_j = \frac{a_{22} a_{11} - a_{21} a_{12}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

Es importante señalar que el algoritmo está diseñado para funcionar para el jugador cuyos pagos se expresan en las filas, por lo que, para el jugador 2, que tiene sus pagos expresados en las columnas, se debe transponer la matriz de pagos antes de realizar los cálculos.

4.2.2. Contexto de un problema

Para esta sección se emplea un ejemplo sin una argumentación practica, diseñado con fines ilustrativos, que se describe con la siguiente matriz de pagos:

$$\begin{array}{ll} \text{J1}\backslash\text{J2} & \begin{array}{cc} c & d \end{array} \\ \begin{array}{l} A \\ B \end{array} & \begin{pmatrix} (-5,7) & (2,17) \\ (9,4) & (5,20) \end{pmatrix} \end{array}$$

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.5.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/ej_algeb.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/ej_algeb.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "ej_algeb" como parámetro.

```
> repMatr(ej_algeb)
```

Salida

Se obtiene una representación visual matricial del juego "ej_algeb" que como se observa, es análoga a la descripción teórica del conflicto.

```
     J1\\J2       c       d
          A (-5,7) (2,17)
          B  (9,4) (5,20)
```

4.2.3. Solución del problema en forma teórica

Como el juego no es de suma cero, es necesario proceder para ambos jugadores.

- Para el jugador 1, se debe aislar sus pagos de la matriz de pago.

$$\begin{matrix} \\ \text{J1} \end{matrix} \begin{matrix} \begin{matrix} \quad & \quad \end{matrix} \\ \begin{pmatrix} -5 & 2 \\ 9 & 5 \end{pmatrix} \end{matrix}$$

- Ahora se calcula su perfil de estrategia conservadora

$$x^* = \frac{a_{22} - a_{21}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

$$\implies x^* = \frac{5-9}{-5-9-2+5} = \frac{-4}{-11} = 0.3636364$$

$$\therefore x_1^* = (\frac{4}{11}, \frac{6}{11})$$

- Ahora se calcula el máximo asegurable del jugador 1

$$v_1 = \frac{a_{22}a_{11} - a_{21}a_{12}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

$$\implies v_1 = \frac{(5)(-5) - (2)(9)}{-5-9-2+5} = \frac{-25-18}{-11} = \frac{-43}{-11} = 3.909091$$

$$\therefore v_1 = 3.909091$$

- Para el jugador 2, se debe aislar sus pagos de la matriz de pago.

$$\text{t(J2)} \qquad \begin{pmatrix} 7 & 17 \\ 4 & 20 \end{pmatrix}$$

- Como el jugador 2 tiene sus pagos expresados en las columnas, se debe transponer la matriz de pagos.

$$\text{J2} \qquad \begin{pmatrix} 7 & 4 \\ 17 & 20 \end{pmatrix}$$

- Se calcula su perfil de estrategia conservadora

$$x^* = \frac{a_{22} - a_{21}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

$$\implies x^* = \frac{20-17}{7-17-4+20} = \frac{3}{6} = 0.5$$

$$\therefore x_1^* = (\frac{1}{2}, \frac{1}{2})$$

- Ahora se calcula el máximo asegurable del jugador 2

$$v_2 = \frac{a_{22}a_{11} - a_{21}a_{12}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

$$\implies v_1 = \frac{(20)(7) - (17)(4)}{7 - 17 - 4 + 20} = \frac{140 - 68}{6} = \frac{72}{6} = 12$$

$$\therefore v_2 = 12$$

4.2.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.4.2. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/EM_Algebraico.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/EM_Algebraico.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue.

Entrada

Para obtener la *estrategia conservadora* del jugador 1 se usa la función *estrConsAlgebr*, con el argumento tipo data frame "ej_algeb" y el argumento tipo entero "1" para los parámetros "juego" y "numero de jugador"respectivamente.

```
> estrConsAlgebr(ej_algeb,1)
```

Salida

Se obtiene el perfil de la *estrategia conservadora* del jugador 1. La primer columna indica las posibles estrategias del jugador 1, y la segunda columna muestra sus pesos asociados.

```
      X         p
A 0.3636364
B 0.6363636
```

Entrada

Para obtener el *valor del juego (máximo asegurable en EM)* del jugador 1 se usa la función *valorJuegAlgebr*, con el argumento tipo data frame "ej_algeb" y el argumento tipo entero "1" para los parámetros "juego" y "numero de jugador"respectivamente.

```
> valorJuegAlgebr(ej_algeb,1)
```

Salida

Se obtiene el *valor del juego* del jugador 1, que indica de forma numérica el pago esperado que tendrá el jugador si juega con la estrategia conservadora.

```
      3.909091
```

Entrada

Para obtener la *estrategia conservadora* del jugador 2 se usa la función *estrConsAlgebr*, con el argumento tipo data frame "ej_algeb" y el argumento tipo entero "2" para los parámetros "juego" y "numero de jugador"respectivamente.

```
> estrConsAlgebr(ej_algeb,2)
```

Salida

Se obtiene el perfil de la *estrategia conservadora* del jugador 2. La primer columna indica las posibles estrategias del jugador 2, y la segunda columna muestra sus pesos asociados. Se puede observar que ambas estrategias tienen el mismo peso.

```
      X   p
c 0.5
d 0.5
```

Entrada

Para obtener el *valor del juego (máximo asegurable en EM)* del jugador 2 se usa la función *valorJuegAlgebr*, con el argumento tipo data frame "ej_algeb" y el argumento tipo entero "2" para los parámetros "juego" y "numero de jugador"respectivamente.

```
> valorJuegAlgebr(ej_algeb,2)
```

Salida

Se obtiene el *valor del juego* del jugador 2, que indica de forma numérica el pago esperado que tendrá el jugador si juega con la estrategia conservadora.

```
1        12
```

4.2.5. Otros problemas para ampliar

Se invita a probar los programas con algunos otros juegos, e identificar alguno donde el cálculo de la estrategia conservadora no esté en el intervalo [0,1], esto con el fin de comprobar que la rutina va a devolver la estrategia pura adecuada según la teoría.

Capítulo 5

Antagonismo

5.1. Estrategias puras

5.1.1. Definición

En general, cuando se habla de antagonismo, se hace referencia a que existen varios intereses que persiguen finalidades opuestas en algún sentido.
Un juego antagónico se caracteriza como aquel en donde la suma de los máximos asegurables de los jugadores es igual a la máxima suma de ganancias de los jugadores de todos los perfiles de estrategias del juego. Es decir, si coincide la máxima ganancia conjunta, con la suma de los máximos asegurables de todos los jugadores (Véase [A5]).

La noción anterior se define formalmente a continuación, partiendo de las definiciones iniciales (Véase 1.1.2.1 , 1.1.3.1 y 1.1.3.2).

Definición 5.1.1.1. .
Se dice que el juego $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$ *es antagónico, si*

$$\sum_{j\in N} v'_j = Max$$

donde Max es la máxima ganancia conjunta,

$$Max = \max_{\sigma\in D} \sum_{j\in N} \varphi_j(\ \sigma)$$

5.1.2. Contexto de un problema

Aliados vs Japón

En febrero de 1943, George Kenney, el comandante en jefe de las fuerzas aéreas aliadas en el sur del Pacífico, tuvo que enfrentarse al siguiente problema (Véase [A2]).

Los japoneses estaban intentando reforzar su ejército en Nueva Guinea, y para ello tenían que decidirse por transportar sus fuerzas por dos rutas alternativas. Podían navegar o bien al norte de Nueva Bretaña, donde el tiempo era lluvioso, o por el sur de la misma, en el que el tiempo era generalmente bueno. En cualquiera de ambos casos el viaje duraba tres días.

Los japoneses deseaban que su flota estuviese expuesta el menor tiempo posible al bombardeo de sus enemigos. El general Kenney, por su parte, tenía que decidir dónde debía concentrar el grueso de sus aviones bombarderos. Ambos tienen que decidir si irán por la norte Sur o Norte.

El conflicto anterior es un juego que puede describirse con la siguiente matriz de pagos:

$$\begin{array}{cc} \text{A}\backslash\text{J} & \begin{array}{cc} Norte & Sur \end{array} \\ \begin{array}{c} Norte \\ Sur \end{array} & \begin{pmatrix} (2,-2) & (2,-2) \\ (1,-1) & (3,-3) \end{pmatrix} \end{array}$$

** Donde A significa "Aliados" y J "Japoneses"*

** Los valores representan el resultado de las combinaciones de las decisiones de ambos bandos. Este resultado significa el número de días de bombardeo, por ello siempre es positivo para A y siempre es negativo para J.*

** Ejemplo del libro de Morton D. Davis "Introducción a la Teoría de Juegos" (1972).*

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.6.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
1 "C:/MisDocumentos/j_guerra.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_guerra.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "j_guerra" como parámetro.

```
> repMatr(j_guerra)
```

Salida

Se obtiene una representación visual matricial del juego "empresasTec" que como se observa, es análoga a la descripción teórica del conflicto.

```
   J1\\J2   Norte     Sur
    Norte (2,-2) (2,-2)
      Sur (1,-1) (3,-3)
```

5.1.3. Solución del problema en forma teórica

Primero se obtiene la *máxima ganancia conjunta* del juego.

$$Max = \max_{\sigma \in D} \sum_{j \in N} \varphi_j(\ \sigma)$$

$$\implies Max = \text{máx}\{\sum_{j \in N} \varphi_j(\ Norte, Norte)\ , \sum_{j \in N} \varphi_j(\ Norte, Sur)\ ,$$

$$\sum_{j \in N} \varphi_j(\ Sur, Norte)\ , \sum_{j \in N} \varphi_j(\ Sur, Sur)\ \}$$

$$\implies Max = \text{máx}\{(\ 2+(\ -2)\ )\ , (\ 2+(\ -2)\ )\ , (\ 1+(\ -1)\ )\ , (\ 3+(\ -3)\ )\ \}$$

$$\implies Max = \text{máx}\{0, 0, 0, 0\}$$

$$\therefore Max = 0$$

Ahora, se obtendrá la *suma de los máximos asegurables para todos los jugadores en el juego.*
El máximo asegurable para los "Aliados" se obtiene a continuación.

$$\varphi_A(\ \sigma|\hat{\sigma}^A = \text{“Norte”})\ = \{2,2\} \geq 2, \forall \sigma \in D$$

$$\therefore\ 2 \text{ es el asegurable para la estrategia “Norte”.}$$

$$\varphi_A(\ \sigma|\hat{\sigma}^A = \text{“Sur”})\ = \{1,3\} \geq 1, \forall \sigma \in D$$

$$\therefore\ 1 \text{ es el asegurable para la estrategia “Sur”.}$$

Como consecuencia, se tiene el conjunto de valores asegurables y el máximo asegurable para los "Aliados" es

$$A'_A = \{2,1\}$$

$$\therefore v'_A = max(\ A'_A)\ = 2$$

Ahora, el máximo asegurable para los "Japoneses".

$$\varphi_J(\ \sigma|\hat{\sigma}^J = \text{“Norte”})\ = \{-2,-1\} \geq -2, \forall \sigma \in D$$

$$\therefore\ \text{-2 es el asegurable para la estrategia “Norte”.}$$

$$\varphi_2(\ \sigma|\hat{\sigma}^1 = \text{“Sur”})\ = \{-2,-3\} \geq -3, \forall \sigma \in D$$

$$\therefore\ \text{-3 es el asegurable para la estrategia “No Competir”.}$$

El máximo asegurable para los "Japoneses" se obtiene a continuación.

$$A'_J = \{-2,-3\}$$

$$\therefore v'_J = max(\ A'_J)\ = -2$$

Después de lo anterior ya se tiene que

$$\sum_{j \in N} v'_j = v'_A + v'_J = 2 + (\ -2) = 0$$

Finalmente, se puede concluir

$$0 = \sum_{j \in N} v'_j = Max$$

$$\therefore \text{El juego es antagónico.}$$

5.1.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.5.1. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/Antagonismos.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/Antagonismos.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

También es importante mencionar que puede ser que para este script se ocupen algunas de las funcionalidades de los otros scripts que se ocupan en los capítulos anteriores. Se recomienda cargarlos también.

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue.

Entrada

Para obtener la *máxima ganancia conjunta del juego* se usa la función *Max*, con el argumento tipo data frame "j_guerra" para el parámetro "juego".

```
> Max(j_guerra)
```

Salida

Se obtiene el valor numérico de la *máxima ganancia conjunta del juego*.

```
     0
```

Entrada

Para determinar si el juego es *antagónico* se usa la función *antagonico*, con el argumento tipo data frame "j_guerra" para el parámetro "juego".

```
> antagonico(j_guerra)
```

Salida

Se obtiene el resultado booleano de la evaluación. Se observa que este juego sí es antagónico.

```
1 TRUE
```

5.1.5. Otros problemas para ampliar

El juego que sirvió de ejemplo para exhibir las capacidades de los programas es particular, ya que aparte de ser antagónico, es de suma cero. Partiendo de esto, se puede usar y se recomienda experimentar con el código generado con cualquier otro tipo de juego no necesariamente suma cero o antagónico.

Se espera y se sugiere como ejercicio que si el lector desea realizar un programa que compruebe si un juego es suma cero, pueda lograrlo modificando muy poco el código para obtener la máxima ganancia conjunta.

Por otro lado, se recomienda experimentar con el juego contenido en D.15, que se incluye en el "D.Recopilación de algunos juegos ", ya que es un juego antagónico donde los jugadores tienen más estrategias.

Por supuesto, también se sugiere sobre estos juegos estudiar su equilibrio de Nash, estrategia conservadora, dominancias, etc.

Este breve apartado es importante ya que existen diversos teoremas en los cuales, dado que un juego es antagónico, se pueden afirmar otras características y resultados sobre ese juego. Si además, el juego es bipersonal y/o suma cero, se consigue una cantidad aún mayor de estos teoremas.

El teorema que resulta más relevante para los fines de este trabajo, inaugura el siguiente capitulo, ya que enlaza el antagonismo con la existencia de puntos silla.

Teorema 5.1.5

Si un juego $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$ *es antagónico, entonces tiene punto silla. En particular, cualquier perfil formado por estrategias conservadoras es punto silla del juego.*

5.2. Estrategias mixtas

5.2.1. Definición

El antagonismo en estrategias mixtas hereda su concepto, argumentación y definición del antagonismo en estrategias puras.
Recordando; un juego antagónico se caracteriza como aquel en donde la suma de los máximos asegurables de los jugadores es igual a la máxima suma de ganancias de los jugadores de todos los perfiles de estrategias del juego. Es decir, si coincide la máxima ganancia conjunta, con la suma de los máximos asegurables de todos los jugadores.

Definición 5.2.1.1. .
Se dice que el juego $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$ *es antagónico en estrategias mixtas, si*

$$\sum_{j\in N} v'_j = Max$$

donde Max es la máxima ganancia conjunta,

$$Max = \max_{\sigma\in D} \sum_{j\in N} \varphi_j(\ \sigma)$$

y v_j *es el valor del juego para cada jugador.*

5.2.2. Contexto de un problema

Volados

Dos hermanos quieren apostar las monedas de un peso que han recibido por su domingo. El primero saca una moneda y la coloca sobre la mesa cubriéndola con la mano, le pide a su hermano que adivine cuál es la posición en que la puso, ¿el águila hacia arriba o el sol? Si el segundo hermano adivina recibirá el peso, pero si falla tendrá que pagar con uno igual. Esta es, por supuesto, una situación que depende del azar.
La matriz de pago es

$$\begin{array}{cc} \text{H1}\backslash\text{H2} & \begin{array}{cc} \textit{Aguila} & \textit{Sol} \end{array} \\ \begin{array}{c} \textit{Aguila} \\ \textit{Sol} \end{array} & \begin{pmatrix} (-1,1) & (1,-1) \\ (1,-1) & (-1,1) \end{pmatrix} \end{array}$$

** Donde H1 significa "hermano 1" y H2 "hermano 2"*

** Los valores representan los resultados de la apuesta*

En estrategias puras es fácilmente comprobable que el juego no tiene ni equilibrio de Nash, ni es antagónico, pero a continuación se comprobará si en estrategias mixtas es antagónico. Esto es para la situación en que los hermanos decidan repetir el juego de volados muchas veces a lo largo del tiempo, situación que suele suceder a veces hasta que un hermano consiga llevar a "la quiebra.ªl otro.

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.7.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_volados.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_volados.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "j_volados" como parámetro.

```
> repMatr(j_volados)
```

Salida

Se obtiene una representación visual matricial del juego "j_volados" que como se observa, es análoga a la descripción teórica del conflicto.

```
  J1\\J2 aguila    sol
  Aguila (-1,1) (1,-1)
     Sol (1,-1) (-1,1)
```

5.2.3. Solución del problema en forma teórica

Primero se obtiene la *máxima ganancia conjunta* del juego.

$$Max = \max_{\sigma \in D} \sum_{j \in N} \varphi_j(\ \sigma)$$

$$\implies Max = \text{máx}\{\sum_{j\in N} \varphi_j(\ Aguila, aguila)\ , \sum_{j\in N} \varphi_j(\ Aguila, sol)\ ,$$

$$\sum_{j\in N} \varphi_j(\ Sol, aguila)\ , \sum_{j\in N} \varphi_j(\ Sol, sol)\ \}$$

$$\implies Max = \text{máx}\{(\ (\ -1)\ +1)\ , (\ 1+(\ -1)\ )\ , (\ 1+(\ -1)\ )\ , (\ (\ -1)\ +1)\ \}$$

$$\implies Max = \text{máx}\{0, 0, 0, 0\}$$

$$\therefore Max = 0$$

Ahora, se obtendrá de forma algebraica la *suma de los máximos asegurables o valores asegurables* para ambos jugadores en el juego.

Como el juego es de suma cero, ambos jugadores tendrán el mismo máximo asegurable, por lo que basta con resolver para el jugador 1.

- Se aísla la matriz de pago del jugador 1.

$$\begin{matrix} \text{J1} \\ \end{matrix}$$
$$\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

- Se calcula el máximo asegurable del jugador

$$v_1' = \frac{a_{22}a_{11} - a_{21}a_{12}}{a_{11} - a_{21} - a_{12} + a_{22}}$$

$$\implies v_1' = \frac{(-1)(-1) - (1)(1)}{-1 - 1 - 1 - 1} = \frac{1-1}{-4} = \frac{0}{-4} = 0$$

$$\therefore v_1' = 0$$

Después de lo anterior tiene que

$$\sum_{j\in N} v_j' = v_1' + v_2' = 0 + 0 = 0$$

Finalmente, se puede concluir

$$0 = \sum_{j \in N} v'_j = Max$$

$\therefore$ El juego es antagónico en estrategias mixtas

5.2.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.5.2. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/EM_Antagonismo.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/EM_Antagonismo.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

También es importante mencionar que puede ser que para este script se ocupen algunas de las funcionalidades de los otros scripts que se ocupan en los capítulos anteriores. Se recomienda cargarlos también.

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue.

Entrada

Para determinar si el juego es *antagónico en estrategias mixtas* se usa la función *antagonicoEM*, con el argumento tipo data frame "j_volados" para el parámetro "juego".

```
> antagonicoEM(j_volados)
```

Salida

Se obtiene el resultado booleano de la evaluación. Se observa que este juego sí es antagónico en estrategias mixtas.

```
    TRUE
```

5.2.5. Otros problemas para ampliar

El código presentado en esta sección exhibe la variación del código escrito para estrategias puras. Por su parte, el juego exhibido de ejemplo permite mostrar la efectividad del programa. Se sugiere que se pruebe para otros juegos que no sean suma cero, o que en estrategias puras sí sean antagónicos.

Por supuesto, también se sugiere sobre estos juegos estudiar su equilibrio de Nash, puntos sillas, etc. ya que la intención de este breve apartado es que sea útil para estudiar la teoría, ya que existen diversos teoremas en los cuales, dado que un juego es antagónico, se pueden afirmar otras características y resultados sobre ese juego.

A continuación se enuncian dos de los teoremas más relevantes derivados del antagonismo en estrategias mixtas.

Teorema 5.2.5.1

Sea un juego $(N, \{D_j\}_{\mathrm{j}\in\mathrm{N}}, \{\varphi_j\}_{\mathrm{j}\in\mathrm{N}})$ antagónico en estrategias mixtas;

- Si X^* en M es un equilibrio de Nash en estrategias mixtas, entonces

$$E_j(X^*) = V_j$$

- Si X^* en M es un perfil de estrategias conservadoras en estrategias mixtas, entonces

$$E_j(X^*) = V_j$$

Teorema 5.2.5.2

Si un juego $(N, \{D_j\}_{\mathrm{j}\in\mathrm{N}}, \{\varphi_j\}_{\mathrm{j}\in\mathrm{N}})$ es antagónico en estrategias mixtas, entonces tiene punto silla en estrategias mixtas.

Capítulo 6

Punto Silla

6.0.1. Definición

El concepto matemático de punto silla se refiere a los puntos críticos donde una función tiene un máximo local en una dimensión, pero un mínimo local en otra.
Esta intuición se hereda a la teoría de juegos, ya que una estrategia punto silla será aquella donde coincida la mejor (máxima) respuesta del jugador y se minimice su posible pérdida, es decir, aquella estrategia de un juego donde coincida la mejor respuesta del jugador y la estrategia conservadora del mismo o, de una forma global del juego, donde coincida algún equilibrio de Nash con alguna estrategia conservadora del juego (Véase [A5]).

La noción anterior se define formalmente a continuación, partiendo de las definiciones iniciales (Véase 1.1.2.1 , 1.1.3.1 y 1.1.3.2).

Definición 6.0.1.1. *Considérese el juego rectangular* $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$. *Se dice que* $\tilde{\sigma} \in D$ *es punto silla de la funcion de pago del jugador j si,*

$$\varphi_j(\tilde{\sigma}|\sigma^j) \leq \varphi_j(\tilde{\sigma}) \leq \varphi_j(\sigma|\tilde{\sigma}^j) \ , \ \forall \sigma \in D, \forall \sigma^j \in D_j$$

Adicionalmente, se dice que $\tilde{\sigma}^* \in D$ *es punto silla del juego si es punto silla de la función de pago de cada jugador.*

Con un poco de observación puede notarse que la definición se compone de:

- La definición de equilibrio de Nash, si se observan el primer y segundo elemento de la desigualdad.

- La definición de estrategia conservadora, si se observan el segundo y el tercer elemento de la desigualdad.

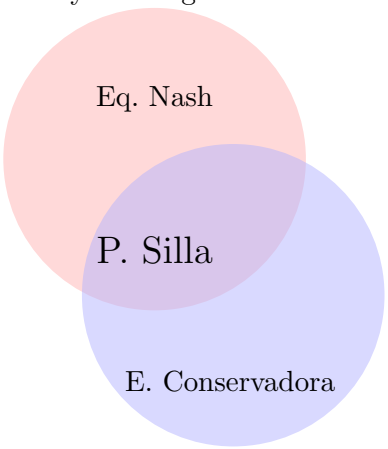
Partiendo de esto, se tiene la siguiente proposición:

Si $\tilde{\sigma}$ *es un punto silla del juego* $(N, \{D_j\}_{j\in N}, \{\varphi_j\}_{j\in N})$*, entonces se cumple:*

- $\tilde{\sigma}$ *es equilibrio de Nash.*
- $\tilde{\sigma}$ *está formada por estrategias conservadoras para cada jugador.*

Este tema se abordará de una forma particular, partiendo del teorema con el que cierra el capitulo anterior y de la última proposición. si bien, NO podemos decir que toda estrategia que sea equilibrio de Nash y estrategia conservadora a la vez será un punto silla, pero sí que si un juego es antagónico, entonces cualquier perfil formado por estrategias conservadoras es punto silla del juego y también equilibrio de Nash.

Aprovechando y evidenciando las ventajas que ofrece la programación, se reutilizaran los algoritmos y códigos respectivos a antagonismo, equilibrio de Nash y estrategias conservadoras.



6.0.2. Contexto de un problema

Para esta sección se vuelve a emplear el ejemplo *Aliados vs. Japón* que se planteó en 5.1.2.

Retomando, el conflicto se describe con la siguiente matriz de pagos:

$$\begin{array}{cc} \text{A}\backslash\text{J} & \begin{array}{cc} Norte & Sur \end{array} \\ \begin{array}{c} Norte \\ Sur \end{array} & \begin{pmatrix} (2,-2) & (2,-2) \\ (1,-1) & (3,-3) \end{pmatrix} \end{array}$$

Partiendo de la forma normal anterior, se puede cargar (Véase A.10) el juego en R con el script de código que se encuentra en D.6.

Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora,

```
"C:/MisDocumentos/j_guerra.R"
```

Entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/j_guerra.R")
```

Una vez cargado el juego, se puede visualizar en forma matricial (Véase 1.2.3) de la siguiente manera:

Entrada

Se declara la función "repMatr" con "j_guerra" como parámetro.

```
> repMatr(j_guerra)
```

Salida

Se obtiene una representación visual matricial del juego "empresasTec" que como se observa, es análoga a la descripción teórica del conflicto.

```
  J1\\J2  Norte    Sur
   Norte (2,-2) (2,-2)
     Sur (1,-1) (3,-3)
```

6.0.3. Solución del problema en forma teórica

Retomando la solución teórica de la sección 2.4.5 se afirma que el juego es antagónico.

Ahora se deben encontrar los perfiles de estrategias conservadoras, que serán los puntos sillas del juego:

Se encontrarán los asegurables para el jugador 1.

$$\varphi_1(\ \sigma|\hat{\sigma}^1 = \text{“Norte”})\ = \{2,2\} \geq 2, \forall \sigma \in D$$

$\therefore$ 2 es el asegurable para la estrategia “Norte”.

$$\varphi_1(\ \sigma|\hat{\sigma}^1 = \text{“Sur”})\ = \{1,3\} \geq 1, \forall \sigma \in D$$

$\therefore$ 1 es el asegurable para la estrategia “Sur”.

Como consecuencia inmediata, se tiene el conjunto de valores asegurables para el jugador 1 y el máximo asegurable.

$$A_1' = \{2,1\}$$

$$\therefore v_1' = max(\ A_1')\ = 2$$

Finalmente, la estrategia conservadora se comprueba por la definición:

$$\varphi_1(\ \sigma|\hat{\sigma}^1 = \text{“Norte”})\ = \{2,2\} \geq 2 = v_1', \forall \sigma \in D$$

$\therefore \hat{\sigma}^1 =$ “Norte” es la estrategia conservadora del jugador 1 para el juego dado.

Se encontrarán los asegurables para el jugador 2.

$$\varphi_2(\ \sigma|\hat{\sigma}^1 = \text{“Norte”})\ = \{-2,-1\} \geq -2, \forall \sigma \in D$$

$\therefore$ -2 es el asegurable para la estrategia “Norte”.

$$\varphi_2(\ \sigma|\hat{\sigma}^1 = \text{“Sur”})\ = \{-2,-3\} \geq -3, \forall \sigma \in D$$

$\therefore$ -3 es el asegurable para la estrategia “Sur”.

Como consecuencia inmediata, se tiene el conjunto de valores asegurables para el jugador 2 y el máximo asegurable.

$$A_2' = \{-2,-3\}$$

$$\therefore v_2' = max(\ A_2')\ = -2$$

Finalmente,la estrategia conservadora se comprueba por la definición:

$$\varphi_2(\ \sigma|\hat{\sigma}^2 = \text{“Norte”})\ = \{-2,-1\} \geq -2 = v_2', \forall \sigma \in D$$

$\therefore \hat{\sigma}^2 =$ “Norte” es la estrategia conservadora del jugador 2 para el juego dado.

Por último:

$\therefore \hat{\sigma} = (Norte, Norte)$, que también es el punto silla del juego.

6.0.4. Solución del problema con programación

Antes de empezar se debe cargar el script de código que se encuentra en C.6. Por ejemplo, si el script se encuentra en la siguiente ubicación en la computadora:

```
"C:/MisDocumentos/PuntoSilla.R"
```

entonces, el script deberá cargarse escribiendo en la linea de comandos de R lo siguiente:

```
source("C:/MisDocumentos/PuntoSilla.R")
```

Además, como algunas de las funcionalidades de la librería "Tidyverse" respaldan el proceso interno de las funciones que se ocupan mas adelante , se debe cargar esta librería con el siguiente comando:

```
library("tidyverse")
```

También es importante mencionar que puede ser que para este script se ocupen algunas de las funcionalidades de los otros scripts que se ocupan en los capítulos anteriores. Se recomienda cargarlos también.

Ahora bien, resolviendo ordenadamente el problema planteado, basta con ejecutar los siguientes comandos para obtener los resultados de interés dependiendo de lo que se investigue. Hay que recordar el el jugador 1 son los "Aliados", mientras que el jugador 2 son los "Japoneses".

Entrada

Para validar que el juego es *antagónico* se usa la función *antagonico*, con el argumento tipo data frame "j_guerra" para el parámetro "juego".

```
> antagonico(j_guerra)
```

Salida

Se obtiene el resultado booleano de la evaluación. Se observa que este juego sí es antagónico y por lo tanto se pueden encontrar sus puntos silla.

```
     TRUE
```

Entrada

Para obtener el *punto silla* del juego para el jugador 1 se usa la función *puntoSilla_j*, con el argumento tipo data frame "j_guerra" y el argumento tipo entero "1" para los parámetros "juego" y "jugador", respectivamente.

```
> puntoSilla_j(j_guerra,1)
```

Salida

Se obtiene el resultado tipo data frame con los puntos sillas del jugador "1" para el juego "j_guerra". Se observa que el punto silla para los "Aliados" consiste en elegir la estrategia "Norte".

```
          S1     S2 PHI1 PHI2
      Norte Norte     2   -2
```

Entrada

Para obtener el *punto silla* del juego para el jugador 2 se usa la función *puntoSilla_j*, con el argumento tipo data frame "j_guerra" y el argumento tipo entero "2" para los parámetros "juego" y "jugador", respectivamente.

```
> puntoSilla_j(j_guerra,2)
```

Salida

Se obtiene el resultado tipo data frame con los puntos sillas del jugador "2" para el juego "j_guerra". Se observa que el punto silla para los "Japoneses" consiste en elegir la estrategia "Norte".

```
          S1     S2 PHI1 PHI2
      Norte Norte     2   -2
```

Entrada

Para obtener el *punto silla* del juego se usa la función *puntoSilla* con el argumento tipo data frame "j_guerra" para el parámetro "juego".

```
> puntoSilla(j_guerra)
```

Salida

Se obtiene el *punto silla del juego*. Como era de esperarse, es aquella donde tanto "Aliados" como "Japoneses" eligen la estrategia "Norte".

```
          S1     S2 PHI1 PHI2
      Norte Norte     2   -2
```

6.0.5. Otros problemas para ampliar

El juego que sirvió de ejemplo permite probar que la rutina es eficaz en juegos antagónicos.

Se sugiere que se prueben las funciones para otros juegos, en especial en

aquellos done haya más de dos jugadores, más de dos estrategias conservadoras, más equilibrios de Nash o bien, donde no haya algún equilibrio de Nash.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

El presente proyecto presentó herramientas de programación, creadas en R por el autor, para estudiar la mayoría de los temas de juegos no cooperativos rectangulares finitos relativos al curso "Teoría de Juegos en Economía" impartido en la Facultad de Ciencias.

Este texto procuró mostrar con agilidad las capacidades de los códigos a través de la exploración y experimentación de juegos a resolver. Los programas generados fueron presentados con apego a la metodología, de forma concisa y acompañados de la teoría necesaria.

También se logró compilar de manera ordenada y simple los códigos, pseudocódigos, y juegos de muestra en el repositorio de Github previsto para ello.

Adicionalmente se debe mencionar que después de cada ejemplo, se hizo la invitación al lector de seguir experimentando con las capacidades de los códigos para ampliar su experiencia. Esta invitación descansa en que los programas obtenidos son capaces de aplicarse a una amplia diversidad de problemas de diferentes complejidades. Para los algoritmos de juegos en estrategias puras se obtuvieron programas de capacidades amplias, que logran resolver problemas n personales donde cada jugador posea m estrategias, siendo n y m números enteros. En el caso de los algoritmos de juegos

en estrategias mixtas se produjeron programas con la capacidad de iterar n veces mostrando los resultados locales de cada iteración, con lo cual ahorran al usuario una cantidad considerable de tiempo para la obtención de resultados en comparación con el desarrollo manual.

Para finalizar; lo obtenido en este trabajo presenta una propuesta de valor para lograr un cambio en el contexto en el que se puede enseñar la materia en el salón de clases, ya que estas nuevas herramientas logran ser un material adecuado y útil para su utilización en el apoyo de la enseñanza. Los alumnos con un perfil inclinado a la programación pueden tener una mejor aproximación a la teoría de juegos y, en contra parte, los alumnos con un perfil más teórico pueden encontrar una mejor aproximación a la programación. En ambos casos pueden fortalecer los conocimientos de la materia y de programación.

A nivel personal, el desarrollo de esta tesis ha significado un grato proceso de madurez en mi aprendizaje. En algunos puntos durante la elaboración de los programas llegué a argüir algunos teoremas o métodos que previamente desconocía. Esto fue profundamente significativo para mí porque evidencia que he adquirido la madurez suficiente para poder aprovechar mi habilidad y conocimiento técnico en crear herramientas que me conduzcan a las preguntas teóricas adecuadas.

Este proyecto me ha exigido mejorar con esmero en competencias que me hacen sentir pleno y que definen quien soy, como son la divulgación efectiva, la docencia y la intensión de contribuir a los demás. Las anteriores razones fueron desde un inicio el motivo de esta tesis.

7.2. Trabajo a futuro

El primer trabajo a futuro que debe enfrentar esta tesis y su contenido es su implementación en el salón de clases, donde se probaría si efectivamente se consiguió su objetivo máximo y dónde se evidenciarán sus deficiencias y áreas de oportunidad. Esto sería aún más enriquecedor si se probara en otras escuelas que impartan la materia aparte de la Facultad de Ciencias o incluso en el escritorio del autodidacta.

El segundo trabajo a futuro es la ampliación del acervo de herramientas

de programación que cubran otros temas de teoría de juegos, así como la ampliación de capacidades de las herramientas aquí presentadas. Todo lo anterior, con miras a crear una librería de R disponible al público en el repositorio en línea oficial de R: CRAN. Algunos otros temas básicos que podrían ser abordados en algún desarrollo futuro son, por ejemplo, las digráficas y los juegos extensivos.

También, puede ser interesante que, partiendo del pseudocódigo de cada tema, los programas puedan adaptarse a otros lenguajes de programación y trabajar en conjunto con las librerías disponibles para estos, como puede ser "AxlRod" de Python, que permite simular juegos evolutivos.

En armonía con todo lo anterior se abre la posibilidad de empezar a usar los programas presentados en conjunto o como complemento con otras herramientas de librerías, ya sean de R o de otro lenguaje de programación, como pueden ser aquellas orientadas a optimización, aprendizaje, o bien, a procesos estocásticos.

En lo referente a la programación; todo código es una manifestación del contexto, recurso y personalidad del programador, por lo que siempre es perfectible. Por ello, es muy claro que todos los códigos presentados en este trabajo están sujetos a mejoras y sofisticación en todos y cada uno de los objetivos de programación. Esto en pro del conocimiento y la eficiencia tanto computacional como humana.

Una mejora inmediata de los programas sería trabajar el código de cada uno para que todos ofrezcan la posibilidad didáctica de mostrar visualmente la resolución algorítmica paso a paso del juego en cuestión, tal como ya se hace con el tema "dominancias". Otra mejora no tan didáctica, sino más bien enfocada en la eficiencia consiste en que casi cualquier función presentada en este material puede ser implementada de forma vectorizada, sin embargo, vale la pena recordar que se ha dado preferencia a que el código propuesto en este material sea de fácil comprensión para un estudiante.

Es importante destacar que donde termina este trabajo empiezan muchos otros. Los códigos presentados son una base utilizable y eficaz para poder dar camino a la experimentación y aplicación a ideas futuras, tanto a la aplicación de problemas concretos como el poder estudiar con detenimiento algunos de los mucho resultados y conjeturas teóricas de la teoría de juegos.

Apéndice A

Algunas nociones de programación

Este apéndice pretende dar orientación sobre aquellos conceptos de programación utilizados en el desarrollo del trabajo. No busca ser una introducción estricta y precisa a la programación o a R, sino más bien una guía para ayudar a aquel que quiera aclarar o continuar estudiando los temas más allá de lo desarrollado en este texto.

A.1. Lenguaje de programación

Se entiende como lenguaje de programación, a un lenguaje formal que especifica una serie de instrucciones para que una computadora produzca diversas clases de datos (Véase [B12]).

Los lenguajes de programación pueden usarse para crear rutinas (programas) que pongan en práctica algoritmos específicos que controlen el comportamiento físico y lógico de una computadora. Los lenguajes de programación están formados por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Existe una gran variedad de lenguajes de programación y su grado de uso depende de diversos factores.

Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.

Un programa permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias.

A grandes rasgos, los lenguajes humanos son sistemas de comunicación que sirven para compartir ideas a otros seres humanos. Por igual, los lenguajes de programación poseen como característica que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.

A.2. Objetivos a cumplir en el código generado para un programa

El código de un programa es el archivo de texto donde se especifican las acciones que componen al programa, escrito según las normas y reglas sintácticas del lenguaje en el que se esté trabajando.

Para que un programa proporcione los mejores resultados su respectivo código debe perseguir una serie de objetivos (Véase [B12]).

Corrección Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases de planeación y delimitación, previas a su implementación. Por ello es muy importante especificar claramente qué debe hacer el programa antes de desarrollarlo y, una vez acabado, compararlo con lo que realmente hace.

Claridad El programa debe ser lo más claro y legible posible, para facilitar el mantenimiento y el trabajo colaborativo. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; de esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que otros programadores puedan continuar el trabajo fácilmente.

Eficiencia El programa, además de realizar aquello para lo que fue creado, debe hacerlo gestionando de la mejor forma posible los recursos que

utiliza. Los recursos hacen referencia al tiempo que tarda en realizar la tarea para la que ha sido creado, la cantidad de memoria que necesita, espacio en disco que utiliza, tráfico de red que genera, etc.

Portabilidad Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea Hardware o Software, diferente a aquella en la que se elaboró. Permite, para empezar, que un programa que se ha desarrollado en una computadora se pueda ejecutar en cualquier otra, incluso aquellas que ocupan un sistema operativo diferente. Esto permite que el programa pueda llegar a más usuarios más fácilmente.

Facilidad de mantenimiento Un programa tiene facilidad de mantenimiento si el esfuerzo requerido para localizar y reparar errores es moderado o mínimo. Este objetivo es, en buena medida, resultado de los otros enlistados.

Flexibilidad Un programa es flexible si el esfuerzo requerido para modificar y/o añadir funcionalidades a una aplicación en funcionamiento es moderado o mínimo. Esto permitirá implementar mejoras y escalamientos a las capacidades del programa.

Reusabilidad Es el grado en que un programa o porción de este es reusable en otras aplicaciones.

A.3. Agrupamiento de código en un programa

Los códigos mostrados en este proyecto siguen un diseño descendente (también conocido como divide y vencerás). Este diseño implica la división de la resolución en varios subprocesos más sencillos que juntos forman la solución completa (Véase [B10]).

Si se busca resolver un problema con programación, la solución completa es la rutina, y a los subprocesos se les llaman funciones y, tal como las funciones matemáticas teóricas, suelen conllevar datos de entrada y de salida. Es decir, necesitan información para llevar a cabo las instrucciones que los componen y al final retornan algún resultado.

Partiendo de los conceptos anteriores, existen paquetes, que son coleccio-

nes de funciones disponibles para su uso, que generalmente están agrupadas según los problemas que tiene como objetivo resolver dicho paquete.

A.4. Pseudocódigo

Una vez que un problema que se pretende programar ha sido analizado, es decir, se ha obtenido el conjunto de datos de entrada, el conjunto de datos de salida esperado, y se ha diseñado un algoritmo que lo resuelva de manera eficiente (lo cual se denomina procesamiento de datos), se debe proceder a la etapa de codificación del algoritmo, pero, para que la solución de un problema pueda ser codificada, es muy importante que previamente se genere una representación del mismo que servirá para orientar al programador en su labor. Una representación algorítmica elemental es el pseudocódigo (Véase [B8]).

Un pseudocódigo es un lenguaje para la representación escrita de un algoritmo, es decir, muestra en forma de texto el flujo lógico a seguir para solucionar un problema. Funciona como un borrador y su realización favorece en conseguir los objetivos de la programación descritos anteriormente.

El pseudocódigo posee una amable sintaxis propia y tiene diversas reglas semánticas y sintácticas. A continuación, se describen brevemente las principales:

1. Alcance del programa: Todo pseudocódigo está limitado por las etiquetas de INICIO y FIN. Dentro de estas etiquetas se deben escribir todas las instrucciones del programa.

2. Palabras reservadas con mayúsculas: Todas las palabras propias del pseudocódigo deben de ser escritas en mayúsculas.

3. Sangría o tabulación: El pseudocódigo debe tener diversas alineaciones para que el código sea más fácil de entender y depurar.

4. Lectura / escritura: Para indicar lectura de datos se utiliza la etiqueta LEER. Para indicar escritura de datos se utiliza la etiqueta ESCRIBIR o IMPRIMIR. La lectura de datos se realiza por defecto desde el teclado. La escritura o impresión de datos se realiza por defecto en la pantalla, que es la salida estándar del sistema.

5. Declaración de variables: la declaración de variables la definen un identificador (nombre), seguido de dos puntos, seguido del tipo de dato. Es posible declarar más de una variable de un mismo tipo de dato utilizando arreglos, indicando después del identificador y entre corchetes la cantidad de variables que se requieren.

6. Se tiene la posibilidad de utilizar operadores aritméticos y lógicos.

7. Para nombrar variables y nombres de funciones se debe hacer uso de la notación de camello también conocida como Sentence Case. En la notación de camello (llamada así porque parecen las jorobas de un camello) los nombres de cada palabra empiezan con mayúscula y el resto se escribe con minúsculas

8. Las estructuras de control de flujo permiten la ejecución condicional y la repetición de un conjunto de instrucciones. Existen 3 estructuras de control: secuencial, condicional y repetitivas o iterativas. Cada una tiene su propia estructura que resulta muy comprensible hasta para el lector inexperto.

9. Una función está constituida por un identificador de función (nombre), parámetros de entrada (de cero a n) y un valor de retorno.

10. Las dos diagonales (//) se utilizan para hacer un comentario y, por tanto, lo que esté después de ellos en la misma línea no es parte del algoritmo y no se toma en cuenta para la ejecución de la rutina. Realizar comentarios sobre un bloque del algoritmo es una buena práctica para detallar sobre el funcionamiento del mismo.

11. Cuando se trabaja con conjuntos de datos estructurados, para elegir un subconjunto de ellos, se utiliza el símbolo $ seguido de una palabra clave que indique el tipo de estructura que se conservará (ELEMENTOS, FILAS o COLUMNAS) y, para finalizar, se debe declarar una cadena que argumente las condiciones lógicas de la subselección.

A.5. Características y relevancia de R

El proyecto de software libre R es un lenguaje y entorno de programación para análisis estadístico y gráfico (Véase [B10]).

Como entorno de programación tiene entre sus objetivos ofrecer un lenguaje simple y eficiente que permita el manejo efectivo de datos y ofrezca herramientas de almacenamiento adecuadas para las exigencias de cada problema.

Además, al ser un paquete estadístico, tiene entre sus prestaciones básicas un amplio abanico de herramientas para la lectura, manipulación, análisis y graficación de datos. Algunas de ellas son: análisis de datos mediante operadores para cálculos sobre arreglos, matrices y/o tablas, tablas cruzadas, reordenamientos de datos, análisis de la varianza (ANOVA), estadística descriptiva, estadística lineal y no lineal, bioestadística, pruebas estadísticas clásicas, análisis de series de tiempo, modelos de regresión, clasificación, fiabilidad, categorización, clustering, y un basto etc.

Otro de los beneficios más grandes que se obtienen al utilizar R es el poder aprovechar un esfuerzo común, ya que el hecho de que R sea un software libre y de código abierto ha permitido ampliar su utilidad a través de la creación constante de nuevos paquetes por la comunidad científica mundial de forma prácticamente gratuita.

Existen miles de paquetes que extienden las capacidades y aplicaciones de R al proveer de funciones que otros usuarios han desarrollado. La mayoría de los paquetes actuales se encuentran instalados en un repositorio en línea llamado CRAN (por sus siglas en inglés, The Comprehensive R Archive Network).

Teniendo en cuenta lo anterior, se debe mencionar la enorme aceptación al uso de software libre entre académicos e investigadores alrededor del mundo. Además de un gran número de entusiastas que están dispuestos a ayudar de forma altruista en numerosos foros de internet.

Otra poderosa ventaja que hereda R del software libre en general es su mantenimiento, ya que por lo regular las versiones se actualizan con mayor frecuencia que la del software propietario que compite, como es el caso de SPSS.

Otras de las características que destacan de R son:

- R también puede usarse como herramienta de cálculo numérico, campo en el que puede ser tan eficaz como otras herramientas diseñadas para ese fin tales como MATLAB/Octave.
- R puede integrarse con distintas bases de datos y existen bibliotecas

que facilitan su utilización desde lenguajes de programación interpretados como Python. Por otro lado, en R se puede utilizar otros lenguajes de programación como C, C++, Fortran, etc.

- Los ambientes de desarrollo integrado para R existen como proyectos externos, como pueden ser editores, los IDEs (Integrate Development Environments) y los GUI (Graphical User Interfaces) que permiten editar, ejecutar y depurar código desarrollado para R. Hay más de 20 proyectos activos, donde el más conocido es RStudio.

- R permite el manejo de datos con diferentes estructuras según las necesidades del problema a resolver. Sin abundar en ellos, los más relevantes y frecuentes son: vector, matrix, list, factor y dataframe.

- Una notable ventaja y característica muy útil que ofrece R sobre otros lenguajes de programación es que para la mayoría de las estructuras de datos se puede trabajar con funciones vectorizadas. Una función vectorizada se aplica no solo en un único valor, sino en todo un vector de valores sin necesidad de realizar ciclos, lo cual se traduce en una considerable mayor eficiencia en tiempo de ejecución y uso de memoria.

Por todo lo anterior, R en la actualidad es, probablemente, el lenguaje de programación más utilizado en investigación por la comunidad estadística. Como consecuencia directa, se ha convertido en la herramienta computacional por excelencia de ciertas comunidades, con el claro ejemplo de la comunidad actuarial mexicana y particularmente, la comunidad de actuarios egresados de la Universidad Nacional Autónoma de México.

A.6. Instalación de R

Para utilizar la paquetería de R, primeramente debe instalarse en la computadora que se usará. Aunque la instalación es sencilla, realizarla de manera correcta es esencial para que R funcione adecuadamente, por ello es importante tener claras algunas especificaciones de nuestro sistema y de nuestras necesidades antes de empezar.

Adicionalmente, para interactuar de forma más simple, interactiva y ágil con R, se recomienda enfáticamente la instalación y uso de un entorno de

desarrollo integrado (IDE por sus siglas en inglés) para R. El más popular, pero no el único, es RStudio.

Por ello, a continuación se adjunta un enlace a una concreta guía en español para la instalación de R y RStudio en los 3 sistemas operativos más difundidos (Windows, GNU/Linux y Mac OS), escrita por Mario Pérez Esteso y propiedad de la Universidad Politécnica de Madrid.

http://www.upm.es/sfs/Rectorado/Gabinete

A.7. Materiales de programación básica con R

Para aprovechar las herramientas y capacidades de R, es imperativo saber programar. Para ello, "La Red Completa de Archivos de R" (CRAN por sus siglas en inglés) en su sitio oficial pone a disposición del público notas de introducción a R en español, en dónde se puede aprender y consultar temas indispensables desde el más básico.

`https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf`

Complementario al documento anterior, en el siguiente enlace se encuentra otra introducción a la programación en R. Este material es de la Universidad Complutense de Madrid y aparte de ser más amable a la vista, se limita mejor a los temas que se ocupan en los códigos contenidos en esta tesis.

`https://www.ucm.es/data/cont/docs/339-2016-09-29-Introduccion%20a%20R_v1617.pdf`

A.8. Data Frames y otras estructuras de Datos

A grandes rasgos una "estructura de datos" es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente (Véase [B10]).

Un data frame es una estructura de datos descrita como una tabla (arreglo bidimensional) en la cual cada columna contiene elementos del mismo tipo de dato, y cada fila contiene un conjunto de elementos que corresponden uno a cada columna.

En los materiales a los que se alude en la sección anterior se puede encontrar el desarrollo de los temas de estructuras de datos, que abordan: vectores, matrices, arreglos, listas, y data frames.

Estos tópicos son de particular importancia para la implementación de los algoritmos que comprende la presente tesis, sobre todo el apartado de "data frames" ya que con esta estructura se modeló y manipulo a los juegos rectangulares.

En el siguiente enlace se pueden visualizar algunas láminas de la Universidad de California en Berkeley que resumen lo más importante de las diversas estructuras de datos y, en especial, de los data frames.

`https://www.stat.berkeley.edu/~nolan/stat133/Fall05/lectures/Lists4.pdf`

A.9. Carga de librerías en R

Las librerías de R son colecciones de funciones y conjuntos de datos desarrollados por algún integrante o equipo de la comunidad de usuarios de R. Las librerías, también conocidas en inglés como "Packages", incrementan las capacidades de R, ya que sus funciones contenidas mejoran viejas funcionalidades o agregan nuevas (Véase [B10]).

Existen varias formas de instalar una librería de R, dependiendo del nivel de desarrollo y disponibilidad de la librería. Las librerías que han conseguido demostrar estabilidad al ser usadas y probadas, y que se encuentran adecuadamente documentadas, se pueden instalar desde el repositorio oficial de R (CRAN) y sus páginas espejo.

La forma más común de instalar alguna librería que se encuentre en el repositorio de CRAN es ejecutar R y escribir en la linea de comandos lo siguiente:

```
1 install.packages("package")
```

Donde la cadena "Package" debe ser sustituida por el nombre de la librería que nos interese. Por ejemplo, si se requiere ocupar una librería llamada "Tidyverse" entonces se tendrá que escribir:

```
install.packages("tidyverse")
```

y esperar a que después de un proceso de instalación, este concluya con un mensaje indicando que se instaló correctamente. El mensaje final luce de la siguiente forma:

```
* DONE (tidyverse)
```

Una vez la librería fue instalada pueden utilizarse todas las herramientas que contiene siempre y cuando se llame a la librería antes de usarla. Para llamar a la librería hay que escribir en la línea de comandos de R lo siguiente y luego dar "enter":

```
library("package")
```

Donde la cadena "Package" debe ser sustituida por el nombre de la librería que nos interese. Por ejemplo, si se requiere ocupar "Tidyverse" entonces se tendrá que escribir:

```
library("tidyverse")
```

Para saber más del uso de librerías en R se deja el siguiente enlace donde se explica a detalle los pormenores de el desarrollo, instalación y mantenimiento de las librerías por parte del equipo de "DataCamp".

`https://www.datacamp.com/community/tutorials/r-packages-guide`

A lo largo de este material se ocupan en escasas ocasiones funciones que no están cargadas predeterminadamente en R, sino en una librería sumamente útil y popular llamada "Tidyverse". En el siguiente enlace, el doctor Joey Stanley de la Universidad de Georgia, hace una excelente introducción y presentación de las funciones más socorridas de dicha librería.

`http://joeystanley.com/downloads/171110-tidyverse_handout.pdf`

A.10. Carga de archivos de códigos a R

Un archivo de código, también conocido como "script", es un archivo de texto que contiene un código escrito como si se fuera a introducir directamente en la linea de comandos (Véase [B10]).

Esto permite que se pueda guardar en uno o varios archivos diferentes las declaraciones de funciones u objetos que se usarán para trabajar, sin tener la necesidad de copiar y pegar código, o peor, de reescribirlo en la línea de comandos.

Un archivo de código se puede cargar de diferentes formas dependiendo su origen. Para cargar archivos que se encunetren en el disco de la computadora donde se está usando R se debe conocer su ubicación en directorio, por ejemplo, si el archivo es "Dominancias.R" y se encuentra en la carpeta "Mis Documentos", un ejemplo de la ubicación sería:

```
1 C:/MisDocumentos/Dominancias.R
```

Y para cargar ese archivo se suele usar, aunque existen otras funciones similares, la función *source()* de la siguiente forma:

```
1 source("C:/MisDocumentos/Dominancias.R")
```

Con ello, ya pdremos ocupar las funciones, variables u objetos que contiene el script "Dominancias.R".

Para saber más del uso y carga de archivos de código en R se deja el siguiente enlace a la página oficial de R donde se explica a detalle.

`https://cran.r-project.org/doc/contrib/Lemon-kickstart/kr_scrpt.html`

Apéndice B

Otros software y paqueterías de teoría de juegos

Actualmente, a pesar de existir un amplio desarrollo de herramientas de programación en general, para el área de teoría de juegos inesperadamente solo se pueden encontrar algunas paqueterías y programas dispersos por foros especializados que si bien ayudan a quienes están inmersos en la teoría de juegos, suelen orientarse a resolver problemas muy particulares y, con frecuencia, con un sustento teórico avanzado o muy específico. Más aún, en el caso de los temas que se encuentran en paqueterías, en su mayoría se reservan a explorar un solo tema abordándolo con diferentes procedimientos o bien, se enfocan en abordar los resultados para los juegos cooperativos.

Entre las herramientas disponibles para los lenguajes de programación con una orientación principal de análisis numérico como Matlab, se encuentra el paquete "MatTuGames" que ofrecen calcular la solución a una cantidad interesante de problemas en teoría de juegos. Sin embargo, se repite el inconveniente de resolver problemas muy específicos, y requerir de un fundamento intermedio o avanzado de teoría de juegos.

Para el caso de Python se pueden encontrar la librería "AxelRod" en la cual se desarrollan temas en específico como la solución del clásico juego del dilema del prisionero, y otras funciones interesantes que distan de ser básicas o de tener un orden y contenido semejante al que se daría en un curso introductorio de nivel licenciatura de la teoría de juegos (Véase [B6]).

Otra librería disponible en Python que se puede encontrar es NashPy (Véase [B9]) que, según su propia descripción, es una librería para el cálculo del equilibrio de Nash en juegos bipersonales por diversos métodos que responden a diferentes recursos de información y diferentes necesidades de información. Acertadamente ese contenido es exacto y estricto, sin abundar en otros temas.

Otras librerías que disponibles son Gambit e Irslib, que, aunque fueron originalmente desarrolladas en el lenguaje de programación "C", actualmente tienen implementación para Python y su objetivo es calcular equilibrios de Nash a través de diferentes algoritmos, perspectivas, recursos de información. Muy semejante a NashPy.

Finalmente, para R existen un par de paquetes, "GameTheory" y "GameTheoryAllocation" respectivamente (Véase [B11]). El primero se concentra en ofrecer herramientas de algoritmos de juegos cooperativos, mientras el segundo se trata de una paquetería cuyas funciones regresan en su mayoría valores booleanos tras evaluar si un determinado valor está o no en el núcleo de un juego, tema que es ajeno a lo que compete al presente trabajo.

Estos excelentes materiales, aunque no satisfacen el objetivo de este trabajo, son una buena fuente de inspiración e ideas sobre el desarrollo de algoritmos para la teoría de juegos.

Apéndice C

Códigos

En este apartado se agrupan los pseudocódigos derivados de las definiciones y algoritmos de cada tema, acompañados de su implementación en R. Estos materiales son el sustento de las funciones que se ocupan a lo largo del texto para resolver problemas de teoría de juegos con tecnologías de la información.

No sobra recordar que estos materiales se encuentran organizados, actualizados y disponibles en el repositorio de Github del proyecto.

Para revisar las convenciones y reglas con las que fueron realizados los pseudocódigos véase [B8]. Por otro lado, si se tiene alguna duda técnica en el desarrollo de los programas en R, se puede encontrar apoyo en el apéndice "A. Algunas nociones de programación" o bien, dirigirse a [B11], [B12] o [B10].

Nota importante: No hay que olvidar que en algunos de los códigos de R se utilizan funciones que están contenidas en la librería para R "Tidyverse". Por esto, es importante que dicha librería sea instalada una vez e importada cada que se disponga a trabajar con los programas siguientes.

C.1. Visualización de los juegos rectangulares

El siguiente código parte de la teoría provista en 1.1.3 , y responde a la necesidad de comprobar si los juegos declarados tienen la expresión visual deseada.

Se trata de una herramienta opcional de apoyo visual.

Función que imprime la representación estratégica de un juego

```
repEstr <- function(juego){
  registros <- data.frame(matrix(NA, nrow = nrow(juego) , ncol
    = 3))
  names(registros) <- c('Forma normal', " ", " ")
  for (i in 1:nrow(juego)) {
    sigma <- "phi("
    phi <- "("
    div_df <- ncol(juego)/2
    for (j in 1:div_df) {
      if (j != div_df) {
        sigma <- paste0(sigma, juego[i,j], "," )
        phi <- paste0(phi, juego[i,div_df+j], "," )
      }else{
        sigma <- paste0(sigma, juego[i,j] )
        phi <- paste0(phi, juego[i,div_df+j] )
      }
      j <- j+1
    }
    registros[i,1] <- (paste0(sigma, ")"))
    registros[i,2] <- (" = ")
    registros[i,3] <- (paste0(phi, ")" ))
    i <- i+1
  }
  registros
}
```

Función que imprime la representación matricial de un juego

```
  if (ncol(juego) != 4 ) {
    print("Este juego no admite forma matricial, ya que no es
    de 2 jugadores.")
    repEstr(juego)
  }else{
    numEst_j1 <- length(unique(juego$S1))+1
    numEst_j2 <- length(unique(juego$S2))+1
    matricial <- data.frame(matrix(NA, nrow = numEst_j1 , ncol
    = numEst_j2 ) )
    matricial[1,1] <- "J1\\J2"
    for (i in 2:numEst_j1) {
      matricial[i,1]<- unique(juego$S1)[i-1]
    }
    for (i in 2:numEst_j2) {
      matricial[1,i]<- unique(juego$S2)[i-1]
    }
    for (i in 2:numEst_j1) {
```

```
      for (j in 2:numEst_j2) {
        phi1 <- juego[(juego$S1 == matricial[i,1]) & (juego$S2
    == matricial[1,j] ), ]$PHI1
        phi2 <- juego[(juego$S1 == matricial[i,1]) & (juego$S2
    == matricial[1,j] ), ]$PHI2
        matricial[i,j] <- paste0("(", phi1,",",phi2 , ")" )
      }
    }
    matricial
  }
}
```

C.2. Dominancias

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 2.0.1 . Se compone de las funciones reduccionPordominancias, dominancias, dominancias_j, juegoReducido, y estrNoDominada.

Función que reduce al máximo un juego iterando la función *dominancias*.

Esta función recibe los parámetros:

1. *juego* : obligatorio de declarar y de tipo dataframe. Corresponde al juego que se quiere reducir.

2. *Visual* : de tipo booleano y con valor predeterminado "True". Sirve para definir si la resolución del juego debe imprimirse paso a paso (True), o si solo nos interesa obtener el resultado (False)

Función que reduce al máximo un juego iterando la función "dominancias".

```
INICIO
  FUNC reduccionPorDominancias(juego:DATAFRAME, Visual:BOOLEANO
    )
    juegoAux:=DATAFRAME
    MIENTRAS ( dim(juego)=1 Y juego!=juegoAux )
      juegoAux = juego
      juego = dominancias(juego, visual)
  FIN MIENTRAS
    RET juego
  FIN FUNC
FIN
```

Función que reduce un juego por sus dominancias.

```
INICIO
  FUNC dominancias(juego:DATAFRAME, visual:BOOLEANO)
    jugadores:= ENTERO
    resultado:= DATAFRAME
    jugadores = ncol(juego)/2
    resultado=juego
    reco, matrDomm EsNoDom, resultado:= DATAFRAME
    PARA i DESDE 1 HASTA jugadores
      reco = recopilado(resultado,i, visual)
      matrDom = matrDominancias(reco, visual)
      EsNoDom = estrNoDominadas(reco, matrDom, visual )
      resultado= juegoReducido(resultado, i, EsNoDom, visual)
    FIN PARA
    SI visual==VERDADERO ENTONCES
    imprimir("EL JUEGO REDUCIDO PARA TODOS LOS JUGADORES ES:")
    imprimir(resultado)
    FIN SI
    RET resultado
  FIN FUNC
FIN
```

Función que obtiene el juego reducido por las dominancias de un jugador.

```
INICIO
  FUNC dominancias_j(juego:DATAFRAME, jugador:ENTERO visual:
    BOOLEANO)
     reco, matrDomm EsNoDom, resultado:= DATAFRAME
   reco = recopilado(resultado,i, visual)
   matrDom = matrDominancias(reco, visual)
   EsNoDom = estrNoDominadas(reco, matrDom, visual )
   resultado = juegoReducido(resultado, i, EsNoDom, visual)
     RET resultado
  FIN FUNC
FIN
```

Función que reduce un juego.

```
INICIO
  FUNC juegoReducido(juego:DATAFRAME, jugador:ENTERO,
    estrNoDominadas:DATAFRAME, visual:BOOLEANO)
     resultado:=DATAFRAME
     resultado = juego JOIN CON estrNoDominadas
     SI visual == VERDADERO ENTONCES
     imprimir(resultado)
```

```
    FIN SI
    RET resultado
  FIN FUNC
FIN
```

Función que obtiene las estrategias no dominadas de un jugador.

```
INICIO
  FUNC estrNoDominadas(recopilado:DATAFRAME, matrDominancias:
    DATAFRAME, visual:BOOLEANO)
    evaluacion[]:= BOOLEANO
    PARA i DESDE 1 HASTA ncol(matrDominancias)
      SI suma(matrDominancias[,i])!=0 ENTONCES
        evaluacion[i] = F
    DE OTRA FORMA
      evaluacion[i] = T
      FIN SI
      estrNoDominadas = (colnames(recopilado))*evaluacion
    SI visual==VERDADERO ENTONCES
    imprimir('Las estrategias no dominadas para el jugador son:
     ')
      imprimir(estrNoDominadas)
      FIN SI
    FIN PARA
    RET estrNoDominadas
  FIN FUNC
FIN
```

Función que obtiene la matriz de dominancias entre las estrategias de un jugador.

```
INICIO
  FUNC matrDominancias(recopilado[]:REAL, visual:BOOLEANO)
    matrizDominancias[][]:= BOOLEANO
    SI visual==VERDADERO ENTONCES
    imprimir('Las dominancias para el jugador son:')
  FIN SI
  PARA i DESDE 1 HASTA longitud(recopilado)
    PARA j DESDE 1 HASTA longitud(recopilado)
      comparaciones = (recopilado[,i] >= recopilado[,j]
          evaluacion = productoCruz(comparaciones)
    FIN PARA
    matrizDominancias[i,j] = 1
    SI visual==VERDADERO ENTONCES
    imprimir(concatenar(colnames(recopilado)[i],"domina a ",
    colnames(recopilado)[j]))
```

```
15          imprimir(concatenar(recopilado[,i]," >= ", recopilado[,
    j]))
16      FIN SI
17    FIN PARA
18      RET matrizDominancias
19    FIN FUNC
20 FIN
```

Función que obtiene la información de un juego y la reordena de forma conveniente para otras funciones.

```
1  INICIO
2    FUNC recopilado(juego:DATAFRAME, jugador:ENTERO visual:
       BOOLEANO)
3       estr:= DATAFRAME
4       separacion := ENTERO
5       phiDeJugador, estrjugadors[] ,estrjugador := CADENA
6       estr <- unique(juego$(COLUMNA:"Estrategias del jugador"))
7       separacion = (ncol(juego)/2)
8     phiDeJugador = colnames(juego$(COLUMNA:"Pagos del jugador"))
9     estrjugadors = colnames(juego$(COLUMNA:"Estrategias de los
        otros jugadores"))
10    estrjugador = colnames(juego$(COLUMNA:"Estrategia del jugador
        "))
11    juegoDeUnaEstr, resulDeUnaEstr, recopilado := DATAFRAME
12    PARA i DESDE 1 HASTA longitud(estr)
13      juegoDeUnaEstr = juego$[FILAS: "La estrategia del jugador
        este en la lista 'estr' "]
14      resulDeUnaEstr = juegoDeUnaEstr$[COLUMNAS: "auqellas cuyo
        nombre este en las listas 'estrjugadors' y 'phiDeJugador' "
15      SI i==1 ENTONCES
16        recopilado <- resulDeUnaEstr
17      DE OTRA FORMA
18        recopilado <- recopilado JOIN CON resulDeUnaEstr
19      FIN SI
20    FIN PARA
21    SI visual==VERDADERO ENTONCES
22      imprimir('Recopilando los perfiles de estrategias que
        conforman los otros jugadores se tiene:')
23      imprimir(recopilado)
24    FIN SI
25    recopilado = recopilado$[COLUMNAS: "Solo los pagos de los
        jugadores "
26      RET recopilado
27    FIN FUNC
28 FIN
```

Código

Función que reduce al máximo un juego iterando la función "dominancias".

```
reduccionPorDominancias <- function(juego, Visual=F){
  i <- 0
  while ((dim(juego)[1]!=1) & !identical(juego,juegoAux) ) {
    i<-i+1
    cat("\n")
    cat("\n")
    print( paste0("#### ITERACION", i, " ####") )
    juegoAux <- juego
    juego <- dominancias(juego, visual=Visual)
  }
  return(juego)
}
```

Función que reduce un juego por sus dominancias.

```
dominancias <- function(juego, visual=F){
  jugadores = ncol(juego)/2
  resultado=juego
  for (i in 1:jugadores) {
    #Esta linea es solo visual
    print( paste0('*** PARA EL JUGADOR:  ', i) )
    reco <- recopilado(resultado,i, visual)
    matrDom <- matrDominancias(reco, visual)
    EsNoDom <- estrNoDominadas(reco, matrDom, visual )
    resultado<- juegoReducido(resultado, i, EsNoDom, visual)
  }
  if (visual==T) {
    cat("\n")
    cat("\n")
    print("EL JUEGO REDUCIDO PARA TODOS LOS JUGADORES ES:")
    print(resultado)
  }
  return(resultado)
}
```

Función que obtiene el juego reducido por las dominancias de un jugador.

```
dominancias_j <- function(juego, jugador, visual=F){
  if (visual==T) {
    cat("\n")
    cat("\n")
    print( paste("PARA EL JUGADOR ", jugador, " TENEMOS:") )
  }
```

```
  reco <- recopilado(juego,jugador, visual)
  matrDom <- matrDominancias(reco, visual)
  EsNoDom <- estrNoDominadas(reco, matrDom, visual )
  resultado<- juegoReducido(juego, jugador, EsNoDom, visual)
  return(resultado)
}
```

Función que reduce un juego.

```
juegoReducido <- function(juego, jugador, estrNoDominadas,
    visual=F){
  juegored <- juego[juego[,jugador] %in% estrNoDominadas,]
  if (visual==T) {
    cat("\n")
    print('El juego reducido para el jugador es: ')
    print(juegored)
  }
  return(juegored)
}
```

Función que obtiene las estrategias no dominadas de un jugador.

```
estrNoDominadas <- function(recopilado,matrDominancias, visual=
    F){
  evaluacion <- c()
  for (i in 1:ncol(matrDominancias)) {
    if (sum(matrDominancias[,i])!=0) {
      evaluacion[i] <- F
    } else{
      evaluacion[i] <- T
    }
  }
  estrNoDominadas <- colnames(recopilado)[evaluacion]
  if (visual==T) {
    cat("\n")
    print('Las estrategias no dominadas para el jugador son: ')
    print(estrNoDominadas)
  }
  return(estrNoDominadas)
}
```

Función que obtiene la matriz de dominancias entre las estrategias de un jugador.

```
matrDominancias <- function(recopilado, visual=F){
  matrizDominancias <- matrix(0, nrow = ncol(recopilado), ncol
    = ncol(recopilado))
  if (visual==T) {
```

```
    cat("\n")
    print('Las dominancias para el jugador son:')
  }
  for (i in 1:length(recopilado)) {
    for (j in 1:length(recopilado)) {
      comparaciones <- as.numeric(recopilado[,i])>=as.numeric(
    recopilado[,j])
      evaluacion <- prod(comparaciones)
      if (evaluacion == 1 & i!=j & matrizDominancias[j,i]==0 )
    {
        matrizDominancias[i,j] <- 1
        if (visual==T) {
          print(paste0(colnames(recopilado)[i]," domina a ",
    colnames(recopilado)[j]))
          print(paste0(recopilado[,i]," >= ", recopilado[,j]))
        }
      }
    }
  }
  if (visual==T) {
    print(matrizDominancias)
  }
  return(matrizDominancias)
}
```

Función que obtiene la información de un juego y la reordena de forma conveniente para otras funciones.

```
recopilado <- function(juego, jugador, visual=F){
  estr <- unique(juego[,jugador])
  separacion <- (ncol(juego)/2)
  phiDeJugador <- colnames(juego)[separacion+jugador]
  estrjugadors <- colnames(juego)[1:separacion]
  estrjugador <- colnames(juego)[jugador]
  estrjugadors <- estrjugadors[which(estrjugadors !=
    estrjugador)]

  for (i in 1:length(estr)) {
    #Resumimos filas
    juegoDeUnaEstr <- juego[which(juego[,jugador] == estr[i]),]
    #Resumimos columnas
    resulDeUnaEstr <- juegoDeUnaEstr[, c(estrjugadors,
    phiDeJugador)]
    # Renonmbramos la columna phi
    colnames(resulDeUnaEstr)[ncol(resulDeUnaEstr)] <- estr[i]
    if (i==1) {
```

```
      recopilado <- resulDeUnaEstr
    }else{
      recopilado <- left_join(recopilado,resulDeUnaEstr, by= c(estrjugadors) )
    }
  }
  if (visual==T) {
    cat("\n")
    print('Recopilando los perfiles de estrategias que conforman los otros jugadores se tiene:')
    print(recopilado)
  }
  #Procedemos a buscar las dominancias, eliminamos los perfiles de estrategias
  recopilado <- recopilado[, separacion:ncol(recopilado)]
  return(recopilado)
}
```

C.3. Equilibrio de Nash

C.3.1. Equilibrio de Nash en estrategias puras

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 3.1.1 . Se compone de las funciones *mejorRespuesta* y *equilibrioNash*.

Función que obtiene la mejor respuesta del jugador.

```
INICIO
  FUNC mejorRespuesta(juego:DATAFRAME ,jugador:ENTERO)
    phiDeJugador := CADENA
    phiDeJugador = juego$(COLUMNA:"phi del jugador")
    estrjugadors := DATAFRAME
    estrjugadors := juego$(COLUMNAS:"estrategias de los otros jugadores")
    resultado := DATAFRAME
    resultado := estrjugadors$(FILAS:"m'aximo valor de phiDeJugador cuando estrjugadors son iguales")
    RET resultado
  FIN FUNC
FIN
```

Función que obtiene el equilibrio de Nash de un juego.

```
INICIO
```

```
  FUNC equilibrioNash(juego:DATAFRAME)
    jugadores := ENTERO
    jugadores = NumeroDeColumnas(juego) / 2
    resultado := DATAFRAME
    PARA i DESDE 1  HASTA jugadores
      SI i==1 ENTONCES
        resultado = mejorRespuesta(juego,1)
      FIN SI
      EN OTRO CASO
        resultado = resultado JOIN CON mejorRespuesta(juego,i)
      FIN DEL EN OTRO CASO
      conj[i] := asegurable(juego,jugador,estr[i])
    FIN PARA
    RET resultado
  FIN FUNC
FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 3.1.4 .

Función que obtiene la mejor respuesta del jugador.

```
mejorRespuesta <- function(juego, jugador, estricta=F){
  separacion <- (ncol(juego)/2)
  phiDeJugador <- colnames(juego)[separacion+jugador]
  estrjugadors <- colnames(juego)[1:separacion]
  estrjugador <- colnames(juego)[jugador]
  estrjugadors <- estrjugadors[which(estrjugadors !=
    estrjugador)]
  resultado <- juego %>%
    group_by_at(estrjugadors) %>%
    filter((!!sym(phiDeJugador)) == max( (!!sym(phiDeJugador)))
      )
  return(resultado)
}
```

Función que obtiene el equilibrio de Nash de un juego.

```
equilibrioNash <- function(juego){
  jugadores <- (ncol(juego)/2)
  for (i in 1:jugadores) {
    if (i==1) {
      resultado <- mejorRespuesta(juego,1)
    }else{
      resultado <- inner_join(resultado, mejorRespuesta(juego,i
    ) )
```

```
    }
  }
  return(resultado)
}
```

C.3.2. Mejor respuesta pura en estrategias mixtas

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 3.2.1 . Se compone de las funciones *juegoMixt*, *respuestasPuras* y *MejorRP*.

Función que modifica un juego para ser usado en estrategias mixtas, calculando el valor esperado de cada estrategia.

```
INICIO
  FUNC juegoMixt(juego:DATAFRAME, X_juego:DATAFRAME)
    resultado := DATAFRAME
    resultado = juego INNER JOIN CON X_juego POR juego$S1=X_
    juego$X
    resultado = resultado INNER JOIN CON X_juego POR juego$S1=X
    _juego$X
    resultado$Xmarg[] ,resultado$E_1[] ,resultado$E_2[] := REAL
    resultado$Xmarg <- resultado$X1 * resultado$X2
    resultado$E_1 <- resultado$Xmarg * resultado$PHI1
    resultado$E_2 <- resultado$Xmarg * resultado$PHI2
    RET resultado
  FIN FUNC
FIN
```

Función que obtiene las respuestas puras de un jugador en un juego de estrategias mixtas.

```
INICIO
  FUNC respuestasPuras(juegoMixt:DATAFRAME, jugador:ENTERO)
    estrJugador[]:= CADENA
    estrJugador = unique(juegoMixt$(COLUMNA:"Estrategias del
    jugador"))
    resultados, juegoAux := DATAFRAME
    juegoAux$valorEstr[] := REAL
    PARA i DESDE 1 HASTA longitud(estrJugador)
      juegoAux = juegoMixt$(FILAS:"La columna de estrategias
    del jugador sea igual a estrJugador[i]")
      juegoAux$valorEstr[] = juegoAux$(FILA:"Peso de la
    estrategia del jugador") * juegoAux$(FILA:"Pagos del jugador
    ")
```

```
10        resultados[i,] = estrJugador[i] UNION CON suma(juegoAux$
      valorEstr)
11      FIN PARA
12      RET resultados
13    FIN FUNC
14 FIN
```

Función que obtiene las mejores respuestas puras de un jugador en un juego de estrategias mixtas.

```
1 INICIO
2   FUNC respuestasPuras(respuestasPuras:DATAFRAME)
3      resultado := DATAFRAME
4      resultado = respuestasPuras
5      resultado = resultado$(FILA:"El pago del jugador sea el ma'
      ximo de la columna de pagos del jugador")
6      resultado = resultado$(COLUMNA:"Estrategias del jugador")
7      RET resultados
8    FIN FUNC
9 FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 3.2.4 .

Función que modifica un juego para ser usado en estrategias mixtas, calculando el valor esperado de cada estrategia.

```
1 juegoMixt <- function(juego, X_juego){
2   # Se hace intersecci'on entre el juego original y el perfil
      de estrategias
3    resultado <- inner_join(juego,X_juego, by=c("S1" = "X") )
4    resultado <- inner_join(resultado,X_juego, by=c("S2" = "X") )
5    names(resultado) <- c('S1', 'S2', 'PHI1', 'PHI2', 'X1', 'X2')
6    # Se obtienen los campos calculados relacionados con el valor
       esperado de cada perfil
7    resultado$Xmarg <- as.numeric(resultado$X1) * as.numeric(
      resultado$X2)
8    resultado$E_1 <- resultado$Xmarg * as.numeric(resultado$PHI1)
9    resultado$E_2 <- resultado$Xmarg * as.numeric(resultado$PHI2)
10   return(resultado)
11 }
```

Función que obtiene las respuestas puras de un jugador en un juego de estrategias mixtas.

```
1 respuestasPuras <- function(juegoMixt, jugador){
```

```
  # Se obtienen las estrategias del jugado
estrJugador<- unique(juegoMixt[,jugador])
  resultados <- data.frame(matrix(NA, nrow = length(estrJugador
    ), ncol = 2), stringsAsFactors=FALSE)
  # Para cada estrategia se calcula el pago esperado
  for (i in 1:length(estrJugador)) {
    juegoAux <- juegoMixt[juegoMixt[,jugador]==estrJugador[i
    ],1:6]
    juegoAux$valorEstr <- as.numeric(juegoAux[,2+jugador]) * as
    .numeric(juegoAux[,ncol(juegoAux)-jugador+1])
    resultados[i,] <- c(estrJugador[i], sum(juegoAux$valorEstr)
     )
  }
  return(resultados)
}
```

Función que obtiene las mejores respuestas puras de un jugador en un juego de estrategias mixtas.

```
mejorRP<- function(respuestasPuras){
  resultado <- respuestasPuras
  resultado <- resultado[resultado[,2]==max(as.numeric(
    resultado[,2])),1]
  return(resultado)
}
```

C.3.3. Juego Ficticio

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 3.3.1, por ello, está ideado para soportar únicamente juegos de dimensión 2x2. Se compone únicamente de la funcion *juegoFicticio. Función que itera t veces con el algoritmo de juego ficticio en un juego 2x2.*

```
INICIO
  FUNC juegoFicticio(juego:DATAFRAME, X_juego:DATAFRAME, t:
    ENTERO)
    aux, aux11, aux21, freq_j1, freq_j2 := DATAFRAME
    aux = juegoMixt(juego,X_juego)
    aux11 = respuestasPuras(aux, 1)
    aux21 = respuestasPuras(aux, 2)
    freq_j1 = aux11 JOIN CON X_juego
    freq_j1 = freq_j1$(COLUMNAS:"Todas menos la columna de
    perfil de estrategias inicial")
```

```
9     freq_j2 = aux21 JOIN CON X_juego
10    freq_j2 = freq_j2$(COLUMNAS:"Todas menos la columna de
      perfil de estrategias inicial")
11    mrp1, mrp2 := CADENA
12    mrp1 = freq_j1$(FILAS:"La frecuencia sea la ma'xima para
      todas las frecuencias del jugador 1")
13    mrp1 = mrp1$(COLUMNA:"La estrategia del jugador 1")
14    mrp2 = freq_j2$(FILAS:"La frecuencia sea la ma'xima para
      todas las frecuencias del jugador 2")
15    mrp2 = mrp2$(COLUMNA:"La estrategia del jugador 2")
16    X_I, X_II, juegoFict := DATAFRAME
17    X_I = [i,
18    freq_j1[VALOR:"Frecuencia de la estrategia del jugador 1 en
       el tiempo i"]/i,
19    1-(freq_j1[VALOR:"Frecuencia de la estrategia del jugador 1
       en el tiepo i"]/i)]
20    X_II = [i,
21    freq_j2[VALOR:"Frecuencia de la estrategia del jugador 2 en
       el tiempo i"]/i,
22    1-(freq_j2[VALOR:"Frecuencia de la estrategia del jugador 2
       en el tiempo i"]/i)]
23    juegoFict = freq_j1 JOIN CON freq_j2 JOIN CON mrp1 JOIN CON
       mrp2
24
25    freq_j1[VALOR:"Frecuencia de la estrategia del jugador 1 en
       el tiempo i"] =
26    freq_j1[VALOR:"Frecuencia de la estrategia del jugador 1 en
       el tiempo i"] +1
27    freq_j2[VALOR:"Frecuencia de la estrategia del jugador 2 en
       el tiempo i"] =
28    freq_j2[VALOR:"Frecuencia de la estrategia del jugador 2 en
       el tiempo i"] +1
29    freq_j1_aux, freq_j2_aux, X_aux := DATAFRAME
30    freq_j1_aux = freq_j1 JOIN CON freq_j1/SUMA(freq_j1)
31    freq_j2_aux = freq_j2 JOIN CON freq_j2/SUMA(freq_j2)
32    X_aux <- freq_j1_aux UNION CON freq_j2_aux
33    X_aux <- X_aux$(COLUMNAS:"Todas menos la columna de perfil
      de estrategias inicial")
34    mrp1 <- mejorRP(aux11)
35    mrp2 <- mejorRP(aux21)
36
37    SI (LONGITUD(mrp1)==1 & LONGITUD(mrp2)==1) ENTONCES
38      PARA i DESDE 2 HASTA t
39        X_I_aux, X_II_aux, juegoFict_aux := DATAFRAME
40        X_I_aux = [i,
```

```
41          freq_j1[VALOR:"Frecuencia de la estrategia del jugador
    1 en el tiempo i"]/i,
42          1-(freq_j1[VALOR:"Frecuencia de la estrategia del
    jugador 1 en el tiepo i"]/i)]
43          X_II_aux = [i,
44          freq_j2[VALOR:"Frecuencia de la estrategia del jugador
    2 en el tiempo i"]/i,
45          1-(freq_j2[VALOR:"Frecuencia de la estrategia del
    jugador 2 en el tiempo i"]/i)]
46          X_I = X_I UNION CON X_I_aux
47          X_II = X_I UNION CON X_II_aux
48          juegoFict_aux = freq_j1 JOIN CON freq_j2 JOIN CON mrp1
    JOIN CON mrp2
49          juegoFict = juegoFict UNION CON juegoFict_aux
50          aux = juegoMixt(juego,X_juego)
51          aux11 = respuestasPuras(aux, 1)
52          aux21 = respuestasPuras(aux, 2)
53          mrp1 <- mejorRP(aux11)
54          mrp2 <- mejorRP(aux21)
55          SI (LONGITUD(mrp1)!=1 & LONGITUD(mrp2)!=1) ENTONCES
56            BREAK
57          FIN SI
58          freq_j1[VALOR:"Frecuencia de la estrategia del jugador
    1 en el tiempo i"] =
59          freq_j1[VALOR:"Frecuencia de la estrategia del jugador
    1 en el tiempo i"] +1
60          freq_j2[VALOR:"Frecuencia de la estrategia del jugador
    2 en el tiempo i"] =
61          freq_j2[VALOR:"Frecuencia de la estrategia del jugador
    2 en el tiempo i"] +1
62          freq_j1_aux, freq_j2_aux, X_aux := DATAFRAME
63          freq_j1_aux = freq_j1 JOIN CON freq_j1/SUMA(freq_j1)
64          freq_j2_aux = freq_j2 JOIN CON freq_j2/SUMA(freq_j2)
65          X_aux <- freq_j1_aux UNION CON freq_j2_aux
66          X_aux <- X_aux$(COLUMNAS:"Todas menos la columna de
    perfil de estrategias inicial")
67        FIN PARA
68      FIN SI
69      RET resultado
70    FIN FUNC
71  FIN
```

Código C.1: Pseudocódigo de Juego Ficticio

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente

script, que se utiliza en 3.3.4 . *Función que itera t veces con el algoritmo de juego ficticio en un juego 2x2.*

```
juegoFicticio <- function(juego, X_juego, t){
  i <- 1
  # Se obtiene el juego mixto
  aux <- juegoMixt(juego,X_juego)
  # Se obtiene las respuestas puras de cada jugador
  aux11 <- respuestasPuras(aux, 1)
  aux21 <- respuestasPuras(aux, 2)
  # Se obtiene la primer frecuencia dle jugador 1
  freq_j1 <- inner_join(aux11, X_juego, by=c('X1'='X') )
  freq_j1 <- freq_j1[,-2]
  freq_j1[,2] <- as.numeric(freq_j1[,2])
  # Se obtiene la primer frecuencia del jugador 2
  freq_j2 <- inner_join(aux21, X_juego, by=c('X1'='X') )
  freq_j2 <- freq_j2[,-2]
  freq_j2[,2] <- as.numeric(freq_j2[,2])
  # Se obtiene la mejor respuesta pura de cada jugador
  # Para la primer iteracion recordemos que fue al azar
  mrp1 <- freq_j1[freq_j1[,2]==max(freq_j1[,2]),1]
  mrp2 <- freq_j2[freq_j2[,2]==max(freq_j2[,2]),1]
  #Guardamos los pesos en un vector
  X_I <- data.frame(i, round( freq_j1[1,2]/sum(freq_j1[,2]) ,
    3) , round( freq_j1[2,2]/sum(freq_j1[,2]) , 3) )
  names(X_I)<-c("t",aux11[,1])
  X_II <- data.frame(i, round( freq_j2[1,2]/sum(freq_j2[,2]) ,
    3) , round( freq_j2[2,2]/sum(freq_j2[,2]) , 3) )
  names(X_II)<-c("t",aux21[,1])
  # Se imprime el juego ficticio
  juegoFict<-data.frame(i, mrp1, paste0('(',X_I[i,2],',',X_I[i
    ,3],')') ,
                        mrp2, paste0('(',X_II[i,2],',',X_II[i
    ,3],')') )
  names(juegoFict)<-c("PeriodoT","I", 'X_I(t)',"II", 'X_II(t)')
  # Recalculamos los valores de la frecuencia, obteniendo la
    sig mrp
  freq_j1[freq_j1[,1]==mejorRP(aux11) ,2] <- freq_j1[freq_j1
    [,1]==mejorRP(aux11) ,2]+1
  freq_j2[freq_j2[,1]==mejorRP(aux21) ,2] <- freq_j2[freq_j2
    [,1]==mejorRP(aux21) ,2]+1
  # se prepara el vector X para las iteraciones poteriores
  freq_j1_aux <- freq_j1 %>% mutate(P=(p/sum(p)))
  freq_j2_aux <- freq_j2 %>% mutate(P=(p/sum(p)))
  X_aux <- rbind(freq_j1_aux , freq_j2_aux )
  X_aux <- X_aux[,-2]
```

```
37  names(X_aux)<-c('X','p')
38  # Se obtiene la mejor respuesta pura de cada jugador
39  mrp1 <- mejorRP(aux11)
40  mrp2 <- mejorRP(aux21)
41
42  # Comienzan las iteraciones para las siguientes repeticiones
43  for (i in 2:t) {
44    #Guardamos los pesos en un vector
45    X_I_aux <- data.frame(i, round( freq_j1[1,2]/sum(freq_j1
      [,2]) ,3) , round( freq_j1[2,2]/sum(freq_j1[,2]) ,3) )
46    names(X_I_aux)<-colnames(X_I)
47    X_I <- rbind(X_I , X_I_aux  )
48
49    X_II_aux <- data.frame(i, round( freq_j2[1,2]/sum(freq_j2
      [,2]) ,3) , round( freq_j2[2,2]/sum(freq_j2[,2]) ,3) )
50    names(X_II_aux)<-colnames(X_II)
51    X_II <- rbind(X_II , X_II_aux  )
52    # Se imprime el juego ficticio
53    juegoFict_aux<-data.frame(i, mrp1, paste0('(',X_I[i,2],',',
      X_I[i,3],')') ,
54                              mrp2, paste0('(',X_II[i,2],',',X_
      II[i,3],')')  )
55    names(juegoFict_aux)<-c("PeriodoT","I", 'X_I(t)',"II", 'X_
      II(t)')
56    juegoFict <- rbind(juegoFict , juegoFict_aux  )
57    # Se obtiene el juego mixto
58    aux <- juegoMixt(juego,X_aux)
59    # Se obtiene las respuestas puras de cada jugador
60    aux11 <- respuestasPuras(aux, 1)
61    aux21 <- respuestasPuras(aux, 2)
62    # Se obtiene la mejor respuesta pura de cada jugador
63    mrp1 <- mejorRP(aux11)
64    mrp2 <- mejorRP(aux21)
65    # Recalculamos los valores de la frecuencia, obteniendo la
      sig mrp
66    freq_j1[freq_j1[,1]==mejorRP(aux11) ,2] <- freq_j1[freq_j1
      [,1]==mejorRP(aux11) ,2]+1
67    freq_j2[freq_j2[,1]==mejorRP(aux21) ,2] <- freq_j2[freq_j2
      [,1]==mejorRP(aux21) ,2]+1
68    # Se prepara el vector X para las iteraciones poteriores
69    freq_j1_aux <- freq_j1 %>% mutate(P=(p/sum(p)))
70    freq_j2_aux <- freq_j2 %>% mutate(P=(p/sum(p)))
71    X_aux <- rbind(freq_j1_aux , freq_j2_aux  )
72    X_aux <- X_aux[,-2]
73    names(X_aux)<-c('X','p')
```

```
74    }
75    return(list(juegoFict,X_I,X_II))
76  }
```

Código C.2: Juego Ficticio

C.3.4. Reajuste de Nash

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 3.4, por ello, está ideado para soportar únicamente juegos de dimensión 2x2. Se compone únicamente de la función *reajusteNash*.

Función que itera t veces con el algoritmo de reajuste de Nash en un juego 2x2.

```
1   FUNC reajusteNash(juego:DATAFRAME ,X_juego:DATAFRAME, t:
      ENTERO )
2     auxColsExtras : DATAFRAME
3     auxColsExtras := juegoMixt(juego,X_juego)
4     auxColsExtras := auxColsExtras$[COLUMNAS: "Columnas 1,2,5,6
      "]
5     X1_ep_compl[], X2_ep_compl[] : REAL
6     X1_ep_compl := 1- auxColsExtras$X1
7     X2_ep_compl := 1- auxColsExtras$X2
8     auxColsExtras := auxColsExtras UNION X1_ep_compl UNION X2_
      ep_compl
9     PARA i desde 1 hasta t
10      X, aux : DATAFRAME
11      X := X_juego$[COLUMNAS:"1 y 1+i"]
12      aux := juegoMixt(juego,X_juego)
13      aux := aux INNER JOIN CON auxColsExtras POR COLUMNAS S1 y
        S2
14      E1_ep[], E2_ep[], E1_ep_compl[], E2_ep_compl[] : REAL
15      E1_ep := aux$PHI1 * aux$X1_ep * aux$X2
16      E2_ep := aux$PHI2 * aux$X2_ep * aux$X1
17      E1_ep_compl := aux$PHI1 * aux$X1_ep_compl * aux$X2
18      E2_ep_compl := aux$PHI2 * aux$X2_ep_compl * aux$X1
19
20      jugador, E_j1 : REAL
21      aux11 : DATAFRAME
22      jugador := 1
23      E_j1 := suma(aux[,7+jugador])
24      aux11 := respuestasPuras(aux,jugador)
```

```
25        aux11[1,2] := suma(aux[,13+jugador])
26        aux11[2,2] := suma(aux[,15+jugador])
27        # Se cambian los nombres de las columnas a "S" y "E_S"
28      C_aux[], C[] : REAL
29        C_aux := aux11$E_s - axu11$E_j1
30        C = SI C_aux<0 ENTONCES 0 DE OTRA FORMA C_aux FIN SI
31        aux111 := axu11 UNION CON C_aux UNION CON C
32        aux111 := aux111 INNER JOIN CON X
33
34        E_j2 : REAL
35        aux21 : DATAFRAME
36        jugador := 2
37        E_j2 := suma(aux[,7+jugador])
38        aux21 := respuestasPuras(aux,jugador)
39        aux21[1,2] := suma(aux[,13+jugador])
40        aux21[2,2] := suma(aux[,15+jugador])
41        # Se cambian los nombres de las columnas a "S" y "E_S"
42        C_aux := aux11$E_s - axu11$E_j2
43        C = SI C_aux<0 ENTONCES 0 DE OTRA FORMA C_aux FIN SI
44        aux211 := aux21 UNION CON C_aux UNION CON C
45        aux211 := aux211 INNER JOIN CON X
46
47        nueva_X[] : REAL
48        nueva_X := (aux111$ante_X + aux111$C) / (1+ aux111$suma_C
      )
49      aux111 := aux111 UNION CON nueva_X
50      nueva_X := (aux211$ante_X + aux211$C) / (1+ aux211$suma_C)
51      aux211 := aux211 UNION CON nueva_X
52      resultado : DATAFRAME
53      resultado := aux211 UNION CON aux111
54      X_juego := X_juego INNER JOIN CON resultado
55      FIN PARA
56      RET resultado
57    FIN FUNC
58  FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 3.4.4 . *Función que itera t veces con el algoritmo de reajuste de Nash en un juego 2x2.*

```
1  reajusteNash <- function(juego, X_juego, t){
2    # Se guardan los perfiles en estrategias puras originales
3    auxColsExtras <- juegoMixt(juego,X_juego)[,c(1,2,5,6)] %>%
4                     mutate(X1_ep_compl = 1- as.numeric(X1) , X2_
       ep_compl = 1-as.numeric(X2)
```

```
5                           ,X1= as.numeric(X1)
6                           ,X2 = as.numeric(X2))
7   colnames(auxColsExtras) <- c('S1', 'S2', 'X1_ep', 'X2_ep', '
      X1_ep_compl', 'X2_ep_compl' )
8   # Se comienza a iterar t veces el proceso
9   for (i in 1:t) {
10    # Se obtienen todos los datos calculados que se usara'n
11    X <- X_juego[,c(1,1+i)]
12    colnames(X) <- c('X',paste0('t',toString(i)) )
13    aux <- juegoMixt(juego,X)
14    aux <- inner_join(aux,auxColsExtras, by=c('S1'='S1','S2'='
      S2'))
15    aux <- aux %>% mutate(E1_ep = as.numeric(PHI1)*X1_ep*as.
      numeric(X2),
16                          E2_ep = as.numeric(PHI2)*X2_ep*as.
      numeric(X1),
17                          E1_ep_compl = as.numeric(PHI1)*X1_ep_
      compl*as.numeric(X2),
18                          E2_ep_compl = as.numeric(PHI2)*X2_ep_
      compl*as.numeric(X1) )
19
20    # Se resuelve para el jugador 1, obtieniendo los
      componentes del cauclo de su X
21    jugador <- 1
22    E_j1 <- sum(aux[,7+jugador])
23    aux11 <- respuestasPuras(aux, jugador)
24    aux11[1,2] <- sum(aux[,13+jugador])
25    aux11[2,2] <- sum(aux[,15+jugador])
26    colnames(aux11) <- c('S','E_S')
27    aux111 <- aux11 %>% mutate(C_aux=as.numeric(E_S)-E_j1 ) %>%
28                        mutate(C=ifelse(C_aux<0,0,C_aux) )
29    suma_C_1 <- sum(aux111$C)
30    aux111 <- aux111 %>% mutate(suma_C=suma_C_1 )
31    colnames(X) <- c('X','ante_X' )
32    aux111 <- inner_join(aux111, X, by=c('S'='X'))
33    aux111$ante_X <- as.numeric( aux111$ante_X )
34
35    #Ahora an'alogo para el jugador 2
36    jugador <- 2
37    E_j2 <- sum(aux[,7+jugador])
38    aux21 <- respuestasPuras(aux, jugador)
39    aux21[1,2] <- sum(aux[,13+jugador])
40    aux21[2,2] <- sum(aux[,15+jugador])
41    colnames(aux21) <- c('S','E_S')
42    aux211 <- aux21 %>% mutate(C_aux=as.numeric(E_S)-E_j2 ) %>%
```

```
                          mutate(C=ifelse(C_aux<0,0,C_aux) )
    suma_C_2 <- sum(aux211$C)
    aux211 <- aux211 %>% mutate(suma_C=suma_C_2 )
    colnames(X) <- c('X','ante_X' )
    aux211 <- inner_join(aux211, X, by=c('S'='X'))
    aux211$ante_X <- as.numeric( aux211$ante_X )

    # Finalemnte se obtiene el perfil de estrategias nuevo
    aux211 <- aux211 %>% mutate(nueva_X= (ante_X + C) / (1+
    suma_C) )
    aux111 <- aux111 %>% mutate(nueva_X= (ante_X + C) / (1+
    suma_C) )
    resultado <- rbind(aux211,aux111)[,c(1,7)]
    colnames(resultado) <- c('X',paste0('t',toString(i)) )

    # Se guarda el nuevo perfil en el data frame de su
    comportamiento histo'rico
    X_juego <- inner_join(X_juego,resultado)

  }
  return(X_juego)
}
```

C.4. Estrategia conservadora y máximo asegurable

C.4.1. Estrategias puras

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 4.1.1. Se compone de las funciones *asegurable*, *conjAsegurable*, *maxAseg* y *estraConserv*.

Función que obtiene el asegurable de una estrategia de un jugador.

```
INICIO
  FUNC asegurable(juego:DATAFRAME ,jugador:ENTERO, estrategia:
    CADENA)
    juegoResumido : DATAFRAME
    juegoResumido := juego$(FILAS:"jugador ELIGE estrategia")
    resultado[] : REAL
    resultado := juegoResumido$(COLUMNA:"pagos para jugador")
    resultado := MIN(resultado)
    RET resultado
```

```
  FIN FUNC
FIN
```

Función que obtiene el asegurable de todas las estrategias de un jugador.

```
INICIO
  FUNC conjAseg(juego:DATAFRAME ,jugador:ENTERO)
    estr[] : CADENA
    # al ser un CONJUNTO sus elementos son unicos, es decir no
    hay elementos repetidos.
    estr := CONJUNTO( juego$(COLUMNA:"estrategias del jugador")
    )
    conj[] : REAL
    PARA i DESDE 1  HASTA LONGITUD(estr)
      conj[i] := asegurable(juego,jugador,estr[i])
    FIN PARA
    res : DATAFRAME
    res$[COLUMNA == 1] := conj[]
    res$[COLUMNA == 2] := estr[]
    RET res
  FIN FUNC
FIN
```

Función que obtiene el máximo asegurable de un jugador.

```
INICIO
  FUNC maxAseg(juego:DATAFRAME ,jugador:ENTERO)
    resultado[] : REAL
    resultado := conjAseg(juego,jugador)$[COLUMNA:"de
    asegurables"]
    resultado := MAX(resultado)
    RET resultado
  FIN FUNC
FIN
```

Función que obtiene las estrategias conservadoras de un jugador.

```
INICIO
  FUNC maxAseg(juego:DATAFRAME ,jugador:ENTERO)
    conjAseg : DATAFRAME
    conjAseg := conjAseg(juego,jugador)
    maxAseg : REAL
    maxAseg := maxAseg(juego,jugador)
    resultado[] : CADENA
    resultado := conjAseg$[FILAS:"pagos sean igual a maxAseg"]
    resultado := resultado$[COLUMNAS:"de estrategias"]
    RET resultado
  FIN FUNC
```

```
FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 4.1.4 .

Función que obtiene el asegurable de una estrategia de un jugador.

```
asegurable <- function(juego, jugador, estrategia){
  #Jugador debe ser numero entero
  juegoResumido <- juego[juego[,jugador]==estrategia,]
  resultado <- juegoResumido[,ncol(juego)/2+jugador]
  resultado <- min(as.numeric(resultado) )
  resultado
  return(resultado)
}
```

Función que obtiene el asegurable de todas las estrategias de un jugador.

```
conjAseg_j <- function(juego, jugador){
  #Todas las estrategias diferentes para el jugador
  estr <- unique(juego[,jugador])
  conj<- c()
  for (i in 1:length(estr)) {
    conj[i] <- asegurable(juego,jugador,estr[i])
  }
  resultado <- as.data.frame(conj)
  resultado$estr <- estr
  return(resultado)
}
```

Función que obtiene el máximo asegurable de un jugador.

```
maxAseg <- function(juego, jugador){
  resultado <- max(as.numeric (as.character(conjAseg_j(juego,
    jugador)$conj) ) )
  return(resultado)
}
```

Función que obtiene las estrategias conservadoras de un jugador.

```
estraConserv <- function(juego, jugador){
  conjAse <- conjAseg_j(juego,jugador)
  resultado <- conjAse[conjAse$conj==maxAseg(juego,jugador) ,]
  resultado <- resultado$estr
  return(resultado)
}
```

C.4.2. Método algebraico

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 4.2, por ello, está ideado para soportar únicamente juegos de dimensión 2x2. Se compone de las funciones *estrConsAlgebr* y *valorJuegAlgebr*.

Función que calcula las estrategias conservadoras de un juego 2x2.

```
INICIO
  FUNC estrConsAlgebr(juego:DATAFRAME ,jugador:ENTERO)
    juegoResumido : DATAFRAME
    juegoResumido := juego$(COLUMNAS:"pagos para el jugador")
    SI jugador==2 ENTONCES
      juegoResumido = TRANSPUESTA(juegoResumido)
  FIN SI
    perfilConse[2] : REAL
    perfilConse[1] := (juegoResumido[2,2]-juegoResumido[2,1] )/
    (juegoResumido[1,1]-juegoResumido[2,1]-juegoResumido[1,2]+
    juegoResumido[2,2] )
    perfilConse[1] := MIN(MAX(perfilConse,0),1)
    perfilConse[2] := 1 - perfilConse[1]
    estrategiasDeJ : DATAFRAME
    estrategiasDeJ := juego$(COLUMNA:"Estrategias S del jugador
    ")
    resultado : DATAFRAME
    resultado := estrategiasDeJ JOIN CON perfilConse
    RET resultado
  FIN FUNC
FIN
```

Función que calcula el valor de un simétrico juego 2x2.

```
INICIO
  FUNC valorJuegAlgebr(juego:DATAFRAME ,jugador:ENTERO)
    juegoResumido : DATAFRAME
    juegoResumido := juego$(COLUMNAS:"pagos para el jugador")
    SI jugador==2 ENTONCES
      juegoResumido = TRANSPUESTA(juegoResumido)
  FIN SI
    resultado : REAL
    resultado := (( juegoResumido[2,2]*juegoResumido[1,1] )-(
    juegoResumido[1,2]*juegoResumido[2,1] ))/(juegoResumido
    [1,1]-juegoResumido[2,1]-juegoResumido[1,2]+juegoResumido
    [2,2] )
    RET resultado
  FIN FUNC
```

```
FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 4.2.4 .

Función que calcula las estrategias conservadoras de un juego 2x2.

```
estrConsAlgebr <- function(juego, jugador){
  # Reducimos el juego a una matriz
  a <- matrix(as.numeric(juego[,2+jugador]),2,2, byrow=T)
  # Recordemos que el jugador dos debe transponerse
  if (jugador==2) {
    a <- t(a)
  }
  # Se realiza el calculo
  x <-  (a[2,2]-a[2,1] )/(a[1,1]-a[2,1]-a[1,2]+a[2,2] )
  # Contenemos donde el perfil no este entre 0 y 1
  x <- min(max(x,0),1)
  # Se calcula el complemento
  xFict <- 1-x
  ### !!! Si el jugador es 2, se debe volver a tranponer?
  # Devolvemos el vector resultado
  X <- unique(juego[,jugador])
  p <- c(x, xFict)
  resultado <- data.frame(X,p)
  return(resultado)
}
```

Función que calcula el valor de un simétrico juego 2x2.

```
valorJuegAlgebr <- function(juego){
  # Basta calcular para el jugador 1
  jugador <- 1
  # Se guarda una matriz con los valores de interes
  a <- matrix(as.numeric(juego[,2+jugador]),2,2, byrow=T)
  # Se calcula y se devuelve
  v <-  (( a[2,2]*a[1,1] )-( a[1,2]*a[2,1] ))/(a[1,1]-a[2,1]-a
    [1,2]+a[2,2] )
  return(v)
}
```

C.5. Antagonismo

C.5.1. Estrategias puras

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 5.1.1. Se compone de las funciones *Max*, *antagonico*.

Función que obtiene la máxima ganancia conjunta de un juego.

```
INICIO
  FUNC Max(juego:DATAFRAME)
     pagosDelJuego : DATAFRAME
     pagosDelJuego := juego$(COLUMNAS:"Pagos para todos los
    jugadores")
     sumaEstr[] : REAL
     sumaEstr := pagosDelJuego$(COLUMNA NUEVA:"Suma de los
    elementos en la fila")
     resultado: REAL
     resultado := MAX(sumaEstr)
    RET resultado
  FIN FUNC
FIN
```

Función que determina si un juego es antagónico o no.

```
INICIO
  FUNC antagonico(juego:DATAFRAME)
     jugadores: REAL
     jugadores:= NCOL(juego)/2
     sumaVi: REAL
     sumaVi:= 0
     PARA i DESDE 1  HASTA LONGITUD(jugadores)
       sumaVi := sumaVi + maxAseg(juego, i)
     FIN PARA
     resultado: BOOLEANO
     resultado: sumaVi = Max(juego)
     RET resultado
  FIN FUNC
FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 5.1.4 .

Función que obtiene la máxima ganancia conjunta de un juego.

```
Max <- function(juego){
  jugadores <- (ncol(juego)/2)
  pagosDelJuego <- sapply(juego[,(jugadores+1):ncol(juego)], as
    .numeric)
  sumaEstr <- rowSums(pagosDelJuego)
  resultado <- max(sumaEstr)
  return(resultado)
}
```

Función que determina si un juego es antagónico o no.

```
antagonico <- function(juego){
  jugadores <- (ncol(juego)/2)
  sumaVi=0
  for (i in 1:jugadores) {
    sumaVi <- sumaVi + maxAseg(juego, i)
  }
  resultado = sumaVi==Max(juego)
  return(resultado)
}
```

C.5.2. Estrategias mixtas

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 5.2.1, por ello, está ideado para soportar únicamente juegos de dimensión 2x2. Se compone únicamente de la función *antagonicoEM*.

Función que determina si un juego es antagónico en estrategias mixtas.

```
INICIO
  FUNC antagonicoEM(juego:DATAFRAME)
    jugadores: REAL
    jugadores:= NCOL(juego)/2
    sumaVi: REAL
    sumaVi:= 0
    PARA i DESDE 1  HASTA LONGITUD(jugadores)
      sumaVi := sumaVi + valorJuegAlgebr(juego, i)
    FIN PARA
    resultado: BOOLEANO
    resultado: sumaVi == Max(juego)
    RET resultado
  FIN FUNC
FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 5.2.4 .

Función que determina si un juego es antagónico en estrategias mixtas.

```
antagonicoEM <- function(juego){
  jugadores <- (ncol(juego)/2)
  sumaVi=0
  for (i in 1:jugadores) {
    sumaVi <- sumaVi + valorJuegAlgebr(juego, i)
  }
  resultado = sumaVi==Max(juego)
  return(resultado)
}
```

C.6. Punto silla

Pseudocódigo

El siguiente pseudocódigo parte de la teoría provista en 6.0.1, por ello, está ideado para soportar únicamente juegos de dimensión 2x2. Se compone de las funciones *puntoSilla_j* y *puntoSilla*.

Función que obtiene los puntos sillas de un jugador en un determinado juego.

```
INICIO
  FUNC puntoSilla_j(juego:DATAFRAME ,jugador:ENTERO)
    estCons[] : cadena
    resultado, estConsAux : DATAFRAME
    estCons := estraConserv(juego, jugador)
  resultado : equilibrioNash(juego)
  estConsAux := dataframe(estCons)
  resultado := resultado INNER JOIN CON estConsAux
    RET resultado
  FIN FUNC
FIN
```

Función que obtiene los puntos sillas de un juego.

```
INICIO
  FUNC puntoSilla(juego:DATAFRAME)
    jugadores : ENTERO
    resultado, resultado_aux : DATAFRAME
```

```
    jugadores := ncol(juego)/2
  resultado := puntoSilla_j(juego, 1)
  PARA i DESDE 2 HASTA jugadores
    resultado_aux := puntoSilla_j(juego, i)
    resultado := resultado INNER JOIN CON resultado_aux
  FIN PARA
  FIN FUNC
FIN
```

Código

El pseudocódigo anterior puede ser implementado en R con el siguiente script, que se utiliza en 6.0.4 .

Función que obtiene los puntos sillas de un jugador en un determinado juego.

```
puntoSilla_j <- function(juego,jugador){
  estCons <- estraConserv(juego, jugador)
  resultado <- as.data.frame(equilibrioNash(juego) )
  estConsAux <- data.frame(t(t(estCons)))
  names(estConsAux) <- paste0('S',jugador)
  resultado <- inner_join(resultado,estConsAux)
  return(resultado)
}
```

Función que obtiene los puntos sillas de un juego.

```
puntoSilla <- function(juego){
  jugadores <- ncol(juego)/2
  resultado <- puntoSilla_j(juego, 1)
  for (i in 2:jugadores) {
    resultado_aux <- puntoSilla_j(juego, i)
    resultado <- inner_join(resultado,resultado_aux)
  }
  return(resultado)
}
```

Apéndice D

Recopilación de algunos juegos

En este apartado se presenta una colección de juegos cuya finalidad es que estén disponibles para experimentar con las capacidades de los programas propuestos en este texto.

Todos ellos se encuentran disponibles en la carpeta "*RecopilacionJuegos*" que está en el repositorio de Github del proyecto.

Antes de empezar es importante tener clara la forma en que se declara un juego. Esta explicación se encuentra en "1.2.2 Declaración de juegos en el entorno de R".

D.1. Juego de las falsas opciones

Este juego es planteado y utilizado como ejemplo en 2.0.2.

El juego está conformado por:

- 3 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: "a1" y "a2" para el jugador uno, "b1" y "b2" para el jugador dos, y "c1" y "c2" para el jugador tres. Las estrategias se declaran en las lineas 3, 4 y 5 como vectores de cadenas.

- Los posibles resultados de la función de pago declarados desde la línea 9 hasta la línea 16.

```
1  j <- 3
2  pe_A <- c('a1', 'a2')
3  pe_B <- c('b1', 'b2')
4  pe_C <- c('c1', 'c2')
5  filas <- length(pe_A) * length(pe_B) * length(pe_C)
6  falsasOpciones <- data.frame(matrix(NA, nrow = filas, ncol = j*
       2), stringsAsFactors=FALSE)
7  names(falsasOpciones) <- c('S1', 'S2', 'S3', 'PHI1', 'PHI2', '
       PHI3')
8  falsasOpciones[1,] <- c('a1', 'b1', 'c1', 1,4,-1 )
9  falsasOpciones[2,] <- c('a1', 'b1', 'c2', 5,6,0 )
10 falsasOpciones[3,] <- c('a1', 'b2', 'c1', 3,2,0 )
11 falsasOpciones[4,] <- c('a1', 'b2', 'c2', 7,6,1 )
12 falsasOpciones[5,] <- c('a2', 'b1', 'c1', 2,8,0 )
13 falsasOpciones[6,] <- c('a2', 'b1', 'c2', 10,8,2 )
14 falsasOpciones[7,] <- c('a2', 'b2', 'c1', 6,4,-2 )
15 falsasOpciones[8,] <- c('a2', 'b2', 'c2', 14,8,-1 )
```

D.2. Juego de las empresas tecnológicas

Este juego es planteado y utilizado como ejemplo en 3.1.2 y en 4.1.2. El juego está conformado por:

- 2 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: “Vetar” y “No Vetar” para el jugador uno, y “Competir” y “No Competir” para el jugador dos. Las estrategias se declaran en las lineas 3 y 4 como vectores de cadenas.
- Los posibles resultados de la función de pago declarados desde la línea 8 hasta la línea 11.

```
1  j <- 2
2  pe_j1 <- c('Vetar', 'No Vetar')
3  pe_j2 <- c('Competir', 'No Competir')
4  filas <- length(pe_j1)*length(pe_j2)
```

```
empresasTec <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(empresasTec) <- c('S1', 'S2', 'PHI1', 'PHI2')
empresasTec[1,] <- c('Vetar', 'Competir', -2, -2)
empresasTec[2,] <- c('Vetar', 'No Competir', 6, -2)
empresasTec[3,] <- c('No Vetar', 'Competir', 4, 4)
empresasTec[4,] <- c('No Vetar', 'No Competir', 5, 3)
```

D.3. Juego de los vecinos corteses

Este juego es planteado y utilizado como ejemplo en 3.2.2 y en 3.3.2. El juego está conformado por:

- 2 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: “Cortes” y “Descortes” para el jugador uno, y “cortes” y “descortes” para el jugador dos. Las estrategias se declaran en las lineas 3 y 4 como vectores de cadenas.
- Los posibles resultados de la función de pago declarados desde la línea 8 hasta la línea 11.
- Adicionalmente, se sugieren dos perfiles de estrategias iniciales para juegos mixtos. En el primero, empezarán con pesos diferentes, mientras que en el segundo ambos jugadores comenzarán siendo corteses.

```
j <- 2
pe_j1 <- c('Cortes', 'Descortes')
pe_j2 <- c('cortes', 'descortes')
filas <- length(pe_j1)*length(pe_j2)
j_vecinos <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(j_vecinos) <- c('S1', 'S2', 'PHI1', 'PHI2')
j_vecinos[1,] <- c('Cortes', 'cortes', -1, -1)
j_vecinos[2,] <- c('Cortes', 'descortes', 0, 2)
j_vecinos[3,] <- c('Descortes', 'cortes', 2, 0)
j_vecinos[4,] <- c('Descortes', 'descortes', -5, -5)
```

Ejemplo utilizado en mejor respuesta pura.

```
X_mrp <- data.frame(matrix(NA, nrow = filas , ncol = 2),
    stringsAsFactors=FALSE)
names(X_mrp) <- c('X', 'p')
X_mrp[1,] <- c('c', 4/7)
X_mrp[2,] <- c('d', 3/7)
X_mrp[3,] <- c('C', 4/7)
X_mrp[4,] <- c('D', 3/7)
```

Ejemplo utilizado en juego ficticio.

```
filas <- length(pe_j1)+length(pe_j2)
X_vecinos <- data.frame(matrix(NA, nrow = filas , ncol = 2),
    stringsAsFactors=FALSE)
names(X_vecinos) <- c('X', 'p')
X_vecinos[1,] <- c('cortes', 1)
X_vecinos[2,] <- c('descortes', 0)
X_vecinos[3,] <- c('Cortes', 1)
X_vecinos[4,] <- c('Descortes', 0)
```

D.4. Juego de ejemplo para Reajuste de Nash

Este juego es planteado y utilizado como ejemplo en 3.4.2.

El juego está conformado por:

- 2 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: “A” y “B” para el jugador uno, y “c” y “d” para el jugador dos. Las estrategias se declaran en las lineas 3 y 4 como vectores de cadenas.
- Los posibles resultados de la función de pago declarados desde la línea 8 hasta la línea 11.
- Adicionalmente, se sugiere un perfil de estrategias inicial para juegos mixtos, donde los jugadores comenzarán en “A” y “c” respectivamente.

```
j <- 2
pe_j1 <- c('A', 'B')
pe_j2 <- c('c', 'd')
filas <- length(pe_j1)*length(pe_j2)
```

```
j_reajusteN <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(j_reajusteN) <- c('S1', 'S2', 'PHI1', 'PHI2')
j_reajusteN[1,] <- c('A', 'c', 2, 4)
j_reajusteN[2,] <- c('A', 'd', 3, -3)
j_reajusteN[3,] <- c('B', 'c', 4, 1)
j_reajusteN[4,] <- c('B', 'd', 5, 2)
```

```
filas <- length(pe_j1)+length(pe_j2)
X_reajusteN <- data.frame(matrix(NA, nrow = filas, ncol = 2),
    stringsAsFactors=FALSE)
names(X_reajusteN) <- c('X', 'p')
X_reajusteN[1,] <- c('A', 1)
X_reajusteN[2,] <- c('B', 0)
X_reajusteN[3,] <- c('c', 1)
X_reajusteN[4,] <- c('d', 0)
```

D.5. Juego de ejemplo para Método Algebraico

Este juego es planteado y utilizado como ejemplo en 4.2.2.

El juego está conformado por:

- 2 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: “A” y “B” para el jugador uno, y “c” y “d” para el jugador dos. Las estrategias se declaran en las lineas 3 y 4 como vectores de cadenas.
- Los posibles resultados de la función de pago declarados desde la línea 8 hasta la línea 11.

```
j <- 2
pe_j1 <- c('A', 'B')
pe_j2 <- c('c', 'd')
filas <- length(pe_j1)*length(pe_j2)
j_algeb <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(j_algeb) <- c('S1', 'S2', 'PHI1', 'PHI2')
j_algeb[1,] <- c('A', 'c', -5, 7)
j_algeb[2,] <- c('A', 'd', 2, 17)
j_algeb[3,] <- c('B', 'c', 9, 4)
j_algeb[4,] <- c('B', 'd', 5, 20)
```

D.6. Juego de los Aliados vs. Japoneses

Este juego es planteado y utilizado como ejemplo en 5.1.2 y en 6.0.2.

El juego está conformado por:

- 2 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: "Norte" y "Sur" para ambos jugadores. Las estrategias se declaran en las lineas 3 y 4 como vectores de cadenas.
- Los posibles resultados de la función de pago declarados desde la línea 8 hasta la línea 11.

```
1  j <- 2
2  pe_j1 <- c('Norte', 'Sur')
3  pe_j2 <- c('Norte', 'Sur')
4  filas <- length(pe_j1)*length(pe_j2)
5  guerra <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
       stringsAsFactors=FALSE)
6  names(guerra) <- c('S1', 'S2', 'PHI1', 'PHI2')
7  guerra[1,] <- c('Norte', 'Norte', 2, -2)
8  guerra[2,] <- c('Norte', 'Sur', 2, -2)
9  guerra[3,] <- c('Sur', 'Norte', 1, -1)
10 guerra[4,] <- c('Sur', 'Sur', 3, -3)
```

D.7. Juego de los Volados

Este juego es planteado y utilizado como ejemplo en 5.2.2.

El juego está conformado por:

- 2 jugadores declarados como un entero en la linea 2.
- 2 estrategias para cada jugador: "Aguila" y "Sol" para el jugador uno, y "aguila" y "sol" para el jugador dos. Las estrategias se declaran en las lineas 3 y 4 como vectores de cadenas.
- Los posibles resultados de la función de pago declarados desde la línea 8 hasta la línea 11.

```
j <- 2
pe_j1 <- c('Aguila', 'Sol')
pe_j2 <- c('aguila', 'sol')
filas <- length(pe_j1)*length(pe_j2)
Volados <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(Volados) <- c('S1', 'S2', 'PHI1', 'PHI2')
Volados[1,] <- c('Aguila', 'aguila', -1, 1)
Volados[2,] <- c('Aguila', 'sol', 1, -1)
Volados[3,] <- c('Sol', 'aguila', 1, -1)
Volados[4,] <- c('Sol', 'sol', -1, 1)
```

Otros juegos, que se declaran de forma análoga a los anteriores, que se pueden encontrar con diferentes variaciones en cualquier bibliografía básica de teoría de juegos (Véase [A3]) son los siguientes:

D.8. Dilema del prisionero

```
j <- 2
pe_j <- c('conf', 'NOconf')
filas <- length(pe_j)^2
prisionero <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(prisionero) <- c('S1', 'S2', 'PHI1', 'PHI2')
prisionero[1,] <- c('conf', 'NOconf', 0, -20)
prisionero[2,] <- c('conf', 'conf', -10, -10)
prisionero[3,] <- c('NOconf', 'NOconf', -2, -2)
prisionero[4,] <- c('NOconf', 'conf', -20, 0)
```

D.9. Disparejo

```
j <- 3
pe_j <- c('arr', 'aba')
filas <- length(pe_j)^3
disparejo <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(disparejo) <- c('S1', 'S2', 'S3', 'PHI1', 'PHI2', 'PHI3')
disparejo[1,] <- c('arr', 'arr', 'arr', 0, 0, 0)
disparejo[2,] <- c('arr', 'arr', 'aba', 0, 0, 1)
disparejo[3,] <- c('arr', 'aba', 'arr', 0, 1, 0)
```

```
disparejo[4,] <- c('arr', 'aba', 'aba', 1, 0, 0)
disparejo[5,] <- c('aba', 'arr', 'arr', 1, 0, 0)
disparejo[6,] <- c('aba', 'arr', 'aba', 0, 1, 0)
disparejo[7,] <- c('aba', 'aba', 'arr', 0, 0, 1)
disparejo[8,] <- c('aba', 'aba', 'aba', 0, 0, 0)
```

Algunos otros juegos, que se declaran de forma análoga a los anteriores, diseñados únicamente para su estudio y sin una motivación de algún contexto real son:

D.10. Juego de ejemplo 1

```
j <- 2
pe_j1 <- c('a', 'b')
pe_j2 <- c('A', 'B', 'C')
filas <- length(pe_j1)*length(pe_j2)
juego_ej1 <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(juego_ej1) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ej1[1,] <- c('a', 'A', 2, 4)
juego_ej1[2,] <- c('a', 'B', 3, -3)
juego_ej1[3,] <- c('a', 'C', 5, 9)
juego_ej1[4,] <- c('b', 'A', 4, 1)
juego_ej1[5,] <- c('b', 'B', 5, 2)
juego_ej1[6,] <- c('b', 'C', 7, -9)
```

D.11. Juego de ejemplo 2

```
j <- 2
pe_j1 <- c('a', 'b')
pe_j2 <- c('A', 'B')
filas <- length(pe_j1)*length(pe_j2)
juego_ej2 <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(juego_ej2) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ej2[1,] <- c('a', 'A', 7, 9)
juego_ej2[2,] <- c('a', 'B', 5, -2)
juego_ej2[3,] <- c('b', 'A', -9, -2)
juego_ej2[4,] <- c('b', 'B', 5, 4)
```

D.12. Juego de ejemplo 3

```
j <- 2
pe_j1 <- c('a', 'b')
pe_j2 <- c('A', 'B')
filas <- length(pe_j1)*length(pe_j2)
juego_ej3 <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(juego_ej3) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ej3[1,] <- c('a', 'A', 4, 3)
juego_ej3[2,] <- c('a', 'B', 9, 2)
juego_ej3[3,] <- c('b', 'A', 5, -7)
juego_ej3[4,] <- c('b', 'B', 3, 2)
```

D.13. Juego de ejemplo 4

```
j <- 2
pe_j1 <- c('a', 'b', 'c', 'd')
pe_j2 <- c('A', 'B', 'C', 'D', 'E')
filas <- length(pe_j1)*length(pe_j2)
juego_ej4 <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(juego_ej4) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ej4[1,] <- c('a', 'A', 5, 9)
juego_ej4[2,] <- c('a', 'B', 3, 2)
juego_ej4[3,] <- c('a', 'C', -3, 3)
juego_ej4[4,] <- c('a', 'D', 5, 3)
juego_ej4[5,] <- c('a', 'E', 2, 5)
juego_ej4[6,] <- c('b', 'A', 4, 3)
juego_ej4[7,] <- c('b', 'B', 2, 2)
juego_ej4[8,] <- c('b', 'C', 5, 3)
juego_ej4[9,] <- c('b', 'D', 2, -3)
juego_ej4[10,] <- c('b', 'E', 12, 2)
juego_ej4[11,] <- c('c', 'A', -1, 1)
juego_ej4[12,] <- c('c', 'B', 3, 8)
juego_ej4[13,] <- c('c', 'C', 2, 7)
juego_ej4[14,] <- c('c', 'D', 5, 9)
juego_ej4[15,] <- c('c', 'E', -9, 2)
juego_ej4[16,] <- c('d', 'A', 3, 1)
juego_ej4[17,] <- c('d', 'B', 11, 2)
juego_ej4[18,] <- c('d', 'C', 3, 4)
juego_ej4[19,] <- c('d', 'D', 5, 9)
juego_ej4[20,] <- c('d', 'E', 9, 4)
```

D.14. Juego de ejemplo 5

```
j <- 2
pe_j1 <- c('N', 'S')
pe_j2 <- c('e', 'o')
filas <- length(pe_j1)*length(pe_j2)
juego_ejemplo5 <- data.frame(matrix(NA, nrow = filas, ncol = j*
    2), stringsAsFactors=FALSE)
names(juego_ejemplo5) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ejemplo5[1,] <- c('N', 'e', 2, 7)
juego_ejemplo5[2,] <- c('N', 'o', 9, 11)
juego_ejemplo5[3,] <- c('S', 'e', -3, -7)
juego_ejemplo5[4,] <- c('S', 'o', 2, 5)
```

```
pe_j1 <- c('N', 'S')
pe_j2 <- c('e', 'o')
filas <- length(pe_j1)+length(pe_j2)
X_ejemplo5 <- data.frame(matrix(NA, nrow = filas, ncol = 2),
    stringsAsFactors=FALSE)
names(X_ejemplo5) <- c('X', 'p')
X_ejemplo5[1,] <- c('N', 1/8)
X_ejemplo5[2,] <- c('S', 7/8)
X_ejemplo5[3,] <- c('e', 3/5)
X_ejemplo5[4,] <- c('o', 2/5)
```

D.15. Juego de ejemplo 6

```
j <- 2
pe_j1 <- c('uno', 'dos', 'tres')
pe_j2 <- c('A', 'B')
filas <- length(pe_j1)*length(pe_j2)
juego_ejemplo6 <- data.frame(matrix(NA, nrow = filas, ncol = j*
    2), stringsAsFactors=FALSE)
names(juego_ejemplo6) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ejemplo6[1,] <- c('uno', 'A', -7, 9)
juego_ejemplo6[2,] <- c('uno', 'B', 0, 1)
juego_ejemplo6[3,] <- c('dos', 'A', 2, 7)
juego_ejemplo6[4,] <- c('dos', 'B', 2, 1)
juego_ejemplo6[5,] <- c('tres', 'A', 3, 7)
juego_ejemplo6[6,] <- c('tres', 'B', 1, 7)
```

```
filas <- length(pe_j1)+length(pe_j2)
X_ejemplo6 <- data.frame(matrix(NA, nrow = filas, ncol = 2),
    stringsAsFactors=FALSE)
```

```
names(X_ejemplo6) <- c('X', 'p')
X_ejemplo6[1,] <- c('uno', 1/3)
X_ejemplo6[2,] <- c('dos', 1/3)
X_ejemplo6[3,] <- c('tres', 1/3)
X_ejemplo6[4,] <- c('A', 3/4)
X_ejemplo6[5,] <- c('B', 1/4)
```

Juego de ejemplo 7

```
j <- 2
pe_j1 <- c('a', 'b')
pe_j2 <- c('A', 'B', 'C')
filas <- length(pe_j1)*length(pe_j2)
juego_ej7 <- data.frame(matrix(NA, nrow = filas, ncol = j*2),
    stringsAsFactors=FALSE)
names(juego_ej7) <- c('S1', 'S2', 'PHI1', 'PHI2')
juego_ej7[1,] <- c('a', 'A', 3, 6)
juego_ej7[2,] <- c('a', 'B', 5, 2)
juego_ej7[3,] <- c('a', 'C', 4, 1)
juego_ej7[4,] <- c('a', 'D', 7, 1)
juego_ej7[5,] <- c('b', 'A', 1, 7)
juego_ej7[6,] <- c('b', 'B', 1, 3)
juego_ej7[7,] <- c('b', 'C', 5, 2)
juego_ej7[8,] <- c('b', 'D', 4, 2)
juego_ej7[9,] <- c('c', 'A', 0, 8)
juego_ej7[10,] <- c('c', 'B', 4, 1)
juego_ej7[11,] <- c('c', 'C', 5, -1)
juego_ej7[12,] <- c('c', 'D', 1, -1)
```

Bibliografía

Bibliografía matemática

[A1] R. Gibbons. *Un primer curso de teoría de juegos.* Antoni Bosch, 1992. España.

[A2] D. Morton. *Game Theory: A Nontechnical Introduction.* Dover Publications, 1983. Estados Unidos.

[A3] P. Zapata Lillo. *Economía, política y otros juegos: una introducción a los juegos no cooperativos.* Las prensas de Ciencias. UNAM., 1997. México.

[A4] Osborne M. y Rubinstein A. *A Course in Game Theory.* The MIT Press, 2011. Estados Unidos.

[A5] E. Ventsel. *Elementos de la teoría de juegos.* MIR, 1977. Rusia.

Bibliografía computacional

[B6] V. Knight. Axelrod python library documentation. `https://axelrod.readthedocs.io/en/stable/`. Consultado: 2019-03-10.

[B7] M. Canty. *Resolving Conflicts with Mathematica.* Elsevier Academic Press, 2000. Alemania.

[B8] J. García Cano, E. y Solano Gálvez. Guía práctica de estudio: Pseudocódigo. http://odin.fi-b.unam.mx/salac/practicasFP/fp_p4.pdf. Consultado: 2018-12-04.

[B9] The Nashpy project developers. A python library for the computation of nash equilibria in two player games. https://nashpy.readthedocs.io/en/stable/. Consultado: 2018-10-04.

[B10] M. Crawley. *The R Book*. Wiley, 2007. Inglaterra.

[B11] J. y Vilella C. Cano-Berlanga, S. y Giménez-Gómez. Enjoying cooperative games: The r package gametheory. https://cran.r-project.org/web/packages/GameTheory/index.html. Consultado: 2018-09-22.

[B12] A. Carillo Ledesma e I. González Rosas. Introducción a la programación. http://132.248.182.159/Herramientas/Lenguajes/IntroduccionALaPro
pdf. Consultado: 2018-11-28.

Diversos

[C13] Martínez Hernández A. y Sánchez Mendiola M. *Formación docente en la UNAM: Antecedentes y la voz de su profesorado*. Coordinación de Desarrollo Educativo e Innovación Curricular, 2019. México. Paginas 415 - 430.