Universidad de Valladolid

Facultad de Ciencias Económicas y Empresariales

Trabajo Fin de Grado

Grado en Administración y Dirección de Empresas

Cálculo y visualización de cinco reglas de reparto para el problema de la bancarrota mediante Python

Presentado por:

Víctor Herrezuelo Paredes

Tutelado por:

Carlos Rodríguez Palmero

Valladolid, julio de 2024

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

RESUMEN

El presente trabajo aborda el problema de la bancarrota, que surge cuando se debe distribuir unos recursos limitados entre varios agentes cuyas reclamaciones son superiores a los recursos disponibles.

Este problema puede resolverse utilizando diferentes reglas de reparto. En este trabajo se analizarán teóricamente las cuatro reglas clásicas: proporcional, igual ganancia, igual pérdida, Talmud, además de una regla adicional denominada regla del orden de llegada.

Se ha diseñado y programado una aplicación informática en Python que permite resolver el problema de la bancarrota utilizando las cinco reglas mencionadas anteriormente. Dicho programa permite además representar gráficamente dicho reparto en función del presupuesto disponible. Se ha usado dicha aplicación para analizar un problema de bancarrota en caso de una reducción en los fondos NextGenEU del Marco Financiero Plurianual, emitidos por la Comisión Europea.

Palabras clave: *problema de bancarrota, reglas de reparto, teoría de juegos, Python, pandas, NextGenEU, Marco Financiero Plurianual*

ABSTRACT

The subject of this work is the bankruptcy problem, also known as claims problem, which arises when limited resources are to be distributed among several agents whose claims exceed the available resources.

This problem can be solved using different division rules. In this paper we will theoretically analyse the four classical rules: proportional, constrained equal gains, constrained equal losses, and Talmud, plus an additional rule called the random arrival rule.

A computer application has been designed and written in Python to solve the bankruptcy problem using the five rules mentioned above. This program can also graphically represent the distribution in virtue of to the available budget. This application has been used to analyse a possible reduction in the NextGenEU funds of the Multiannual Financial Framework, issued by the European Commission.

1. INTRODUCCIÓN

El problema de la bancarrota es un tema de interés en el campo de la economía y la teoría de juegos, pues aborda la distribución de recursos limitados de acuerdo con unos principios determinados entre varios agentes que tienen reclamaciones superiores al bien disponible. La motivación para este trabajo surge de la necesidad de comprender y aplicar diferentes reglas de reparto en situaciones de bancarrota, no solo desde una perspectiva teórica sino también práctica. Se pretende también diseñar una herramienta rápida y manejable que pueda ayudar en el análisis económico de problemas de bancarrota y así también apoyar futuros trabajos de investigación.

La primera sección define formalmente el problema de la bancarrota, incluyendo el enunciado y la notación utilizada a lo largo del trabajo, sentando así las bases para el posterior análisis de las reglas de reparto. A continuación, se describe en detalle las diferentes reglas de reparto que se aplican en problemas de bancarrota, abordando cada una en subsecciones específicas. En concreto, éstas son la **regla de reparto proporcional**, la **regla de igual ganancia**, la **regla de igual pérdida**, **la regla del Talmud o del bien disputado** y la **regla del orden de llegada**.

Parte de este trabajo ha consistido en el diseño y elaboración de un script en el lenguaje de programación Python que permite resolver el problema de la bancarrota según las diferentes reglas. En primer lugar, se explican las herramientas utilizadas y se detalla el porqué de su elección. En segundo lugar, se muestra un manual de instrucciones para poder hacer uso de la aplicación. A continuación, se analiza el código para así analizar los algoritmos utilizados en la aplicación.

Por último, hemos aplicado el script en Python para simular el reparto de los fondos europeos del Marco Financiero Plurianual (MFP) y NextGenerationEU (NextGenEU) utilizando cinco reglas de reparto en un escenario de reducción presupuestaria.

2. EL PROBLEMA DE LA BANCARROTA

2.1. Enunciado y notación

Un *problema de bancarrota* surge cuando la cantidad disponible de un bien perfectamente divisible ha de ser repartida entre un conjunto de agentes $M = \{1, 2, \dots, m\}$ y esta cantidad es insuficiente para satisfacer todas las reclamaciones de los agentes. Denotamos por E al *presupuesto* o cantidad total disponible del bien a repartir, y este presupuesto siempre será positivo $E > 0$. Cada agente reclama una cierta cantidad del presupuesto, que denominaremos *reclamaciones* y denotaremos como c_i. El vector de reclamaciones de

todos los agentes se representa como c $= \{c_1, c_2, \dots, c_m\}$. La reclamación agregada C es la suma de todas las reclamaciones individuales tal que $C = \sum_{i \in M} c_i$.

De esta manera, un problema de bancarrota se representa por el par (E, c). Existe un problema de bancarrota siempre que la reclamación agregada sea superior a la cantidad disponible a repartir, es decir, $C > E$. El *déficit* o *pérdida* colectiva se define como la diferencia entre la reclamación agregada y el presupuesto: $C - E$. La solución de un problema de bancarrota viene determinada por una regla de reparto F, que asigna a cada agente i una cantidad x_i^* del bien a repartir, en función del presupuesto disponible y de las reclamaciones de los agentes. El vector de asignaciones es $x^* = x_1^*, x_2^*, \dots, x_m^* \in \mathbb{R}^m$.

Para que una asignación sea una solución válida del problema de bancarrota, debe cumplir dos condiciones: la racionalidad individual y la eficiencia.

i. La racionalidad individual establece que ningún agente recibe más de lo que reclama y las asignaciones no pueden ser negativas, es decir, $0 \leq x_i^* \leq c_i$ para todo $i \in M$.
ii. La eficiencia exige que la suma de todas las asignaciones sea igual al presupuesto disponible, es decir, $\sum_{i \in M} x_i^* = E$ (Villar, 2005).

En este trabajo se van a analizar cinco soluciones al problema de la bancarrota. Las tres primeras, la de reparto proporcional, la de igual ganancia y la de igual pérdida, se basan en un principio de igualdad, pero se diferencian en la variable que buscan igualar. La regla proporcional iguala las proporciones entre las cantidades obtenidas y las cantidades reclamadas. La regla igualitaria iguala las cantidades obtenidas por los agentes. La regla de igual pérdida, por su parte, iguala las pérdidas sufridas en relación con las reclamaciones presentadas. La cuarta solución, conocida como la regla del Talmud, sigue una lógica distinta: asegura que nadie reciba más de la mitad de lo que reclama si la cantidad disponible es inferior a la mitad del total reclamado, y que nadie pierda más de la mitad de lo que reclama si la cantidad disponible supera la mitad del total reclamado. Estas cuatro primeras reglas constituyen las llamadas *reglas clásicas de reparto* (Villar, 2005). La quinta solución que se va a analizar, la del orden de llegada, calcula las asignaciones que percibirían los agentes en función del orden en que se presentan, hasta que se agota el presupuesto.

3. REGLAS DE REPARTO

A continuación, se van a enunciar y caracterizar las cuatro reglas clásicas de reparto junto con la regla de orden de llegada.

3.1. Regla de reparto proporcional

La *solución proporcional*, atribuida a Aristóteles, es probablemente la forma más conocida y utilizada de resolver problemas de bancarrota. Esta solución distribuye el presupuesto proporcionalmente a las reclamaciones de los agentes (Villar, 2005). Formalmente, la solución proporcional P se define de la siguiente manera: para cada problema (E, C) propone una asignación $P(E, c) = \sigma c$, donde:

$$(1) \qquad \sigma = \frac{E}{C}.$$

Según Aumann y Maschler, es la regla más frecuentemente utilizada para distribuir el presupuesto de manera proporcional a las reclamaciones realizadas por cada agente. En esta regla, el coeficiente de proporcionalidad σ se define dividiendo el presupuesto entre la suma de las reclamaciones. Este coeficiente multiplica cada una de las reclamaciones de los agentes para obtener sus asignaciones. Los valores que toma están comprendidos entre cero y uno, debido a que la reclamación agregada no puede ser inferior al presupuesto.

Presentemos ahora, a modo de ilustración, un ejemplo sencillo de la regla proporcional. Supongamos que el ayuntamiento de un pueblo está repartiendo un fondo de ayuda de emergencia de 1000 euros entre cuatro vecinos que han tenido diferentes niveles de pérdidas debido a una inundación. Las reclamaciones de los vecinos son las siguientes:

- vecino 1: 600 euros,
- vecino 2: 300 euros,
- vecino 3: 200 euros,
- vecino 4: 100 euros.

El vector de reclamaciones es $c = (600,300,200,100)$, y la reclamación agregada C es igual a 1200 euros. El coeficiente de proporcionalidad σ se calcula como:

$$(2) \qquad \sigma = \frac{\mathrm{E}}{\mathrm{C}} = \frac{1000}{1200} = \frac{5}{6} \approx 0{,}833\,.$$

Aplicando la regla de reparto proporcional, la solución a este problema de bancarrota vendrá dada por:

$$(3) \qquad P_{i(E,c)} = \sigma c = 0.833 \times (600,300,200,100) = (500,250,166.67,83).$$

Por lo tanto, el reparto que proporciona esta regla es el siguiente:

- vecino 1 recibe 500 euros,
- vecino 2 recibe 250 euros,
- vecino 3 recibe 166.67 euros,
- vecino 4 recibe 83 euros.

De esta forma, haciendo uso de la regla de reparto proporcional, cada vecino recibiría la misma proporción de su reclamación, es decir, el 83.33% la misma.

3.2. Regla de reparto de igual ganancia

En una solución con regla de reparto de igual ganancia, los agentes involucrados en el problema de bancarrota obtienen la misma cantidad, con la restricción de que ninguno de ellos puede recibir una cuantía superior a la reclamada; en cuyo caso, percibirían el total de su reclamación (Villar, 2005). De esta forma, la regla de reparto de igual ganancia, IG , se define de la siguiente manera:

$$(4) \qquad IG_i(E, c) = \min(\lambda, c_i),$$

siendo λ la solución de la ecuación:

$$(5) \qquad \sum_{i \in M} \min(\lambda, c_i) = E.$$

Formalmente, la solución igualitaria I es aquella solución que, para cada problema (E, c), y para cada $i = 1,2, \dots, n$, propone una asignación dada por $IG_i(E, c) = \min(\lambda, c_i)$.

Retomemos el ejemplo utilizado en la regla anterior. El ayuntamiento de un pueblo está repartiendo un fondo de ayuda de emergencia de 1000 euros entre cuatro vecinos. Sus reclamaciones son $c = (600,300,200,100)$, y la reclamación agregada es por lo tanto $C = 1200$.

Primero, calculamos λ a través de la ecuación:

$$(6) \qquad \sum_{i \in M} \min(\lambda, c_i) = 1000.$$

Resolviendo la ecuación[1] obtenemos que $\lambda = 400$. Aplicando la regla de igual ganancia:

$$(7) \quad IG_1(E,c) = \min(400,600) = 400,$$

$$IG_2(E,c) = \min(400,300) = 300,$$

$$IG_3(E,c) = \min(400,200) = 200,$$

$$IG_4(E,c) = \min(400,100) = 100,$$

De esta forma, usando esta regla de reparto, observamos que el vecino 1 obtiene menos de lo que reclamaba, mientras que los otros tres vecinos no pierden ni ganan nada.

3.3. Regla de igual pérdida

La regla para tratar en este apartado reparte el déficit por igual entre cada agente, bajo la restricción de que ningún agente pierda una cantidad superior a la reclamada (Villar, 2005). De esta forma, la regla de reparto de igual pérdida, IP, se define de la siguiente manera:

$$(8) \qquad IP_i(E,c) = \max(0, c_i - \mu),$$

donde μ es la solución de la ecuación:

$$(9) \qquad \sum_{i \in M} \max(0, c_i - \mu) = E.$$

Formalmente, la solución de igual pérdida IP es aquella que, para cada problema (E,c) y para cada $i = 1,2,\dots,n$, propone la asignación dada por $IP_i(E,c) = \max(0, c_i - \mu)$.

Sigamos ahora el mismo ejemplo del pueblo donde el ayuntamiento tiene que repartir un fondo de ayuda de $E = 1000$ frente a unas reclamaciones de $c = (600,300,200,100)$.

Primero, tendríamos primero que calcular μ resolviendo la ecuación:

$$(10) \qquad \sum_{i \in M} max(0, c_i - \mu) = 1000.$$

Resolviendo la ecuación[2], obtenemos que $\mu = 50$.

[1] Para resolver la ecuación se ha utilizado el método computacional de la bisección. Ver 4.3.1.
[2] Para resolver la ecuación se ha utilizado el método computacional de la bisección. Ver 4.3.1.

Aplicando la regla de igual pérdida:

$$(11) \quad IP_1(E,c) = max(0{,}600 - 50) = 550,$$

$$IP_2(E,c) = max(0{,}300 - 50) = 250,$$

$$IP_3(E,c) = max(0{,}200 - 50) = 150,$$

$$IP_4(E,c) = ma\,x(0{,}100 - 50) = 50.$$

De esta forma, comprobamos que haciendo uso de la regla de igual pérdida aquellos vecinos con reclamaciones menores en valor absoluto pierden una mayor proporción de sus reclamaciones.

3.4. Regla del Talmud o del bien disputado

La regla del Talmud es un procedimiento de reparto diseñado para acomodar las soluciones numéricas que aparecen en el Talmud relativas a la resolución de una serie de problemas prácticos, como herencias. Propuesta por Aumann y Maschler, esta regla se basa en el *Mishna*, que compendia las leyes de tradición oral, y en el *Gemara*, que recoge la interpretación de dichas leyes. La regla del Talmud determina la solución de problemas prácticos utilizando criterios psicológicos que afirman que los agentes se fijan en sus ganancias potenciales cuando el presupuesto es pequeño y se preocupan por sus deudas cuando la cantidad a repartir es grande.

Esta regla se identifica como la aplicación de un principio de protección a los agentes según el cual la pérdida de cada individuo será del mismo tipo que la pérdida social. En particular, nadie obtendrá más de la mitad de su deuda cuando la cantidad disponible sea inferior a la mitad de la deuda agregada, y nadie perderá más de la mitad de su reclamación cuando la cantidad disponible supere la mitad de la reclamación total. Villar y Herrero (2000) demostraron que la regla del Talmud puede ser vista como una combinación de diferentes reglas de reparto, dependiendo de la relación entre la cantidad del bien disponible E y la mitad de la reclamación agregada $\frac{C}{2}$.

Formalmente, la solución del Talmud T es aquella solución que, para cada problema (E,c) y para cada $i = 1,2,\dots,m$ propone la asignación:

$$(12) \qquad T_i(E,c) = \begin{cases} min\,(\frac{c_i}{2}, \lambda), & si\ E \leq \frac{C}{2} \\ max\,(\frac{c_i}{2}, c_i - \mu), & si\ E \geq \frac{C}{2} \end{cases}$$

Donde λ es la solución de la ecuación:

$$\sum_{i \in M} \min\left(\lambda, \frac{c_i}{2}\right) = E, \tag{13}$$

y μ es la solución de la ecuación:

$$\sum_{i \in M} \max\left(\frac{c_i}{2}, c_i - \mu\right) = E. \tag{14}$$

Nótese que la definición de λ y μ es diferente a la usada en las reglas de igual ganancia e pérdida.

La regla del Talmud puede identificarse como la aplicación de un principio de protección a los agentes según el cual la pérdida de cada individuo será del mismo tipo que la pérdida social. En particular, nadie obtendrá más de la mitad de su deuda cuando la cantidad disponible sea inferior a la mitad de la deuda agregada, y nadie perderá más de la mitad de su reclamación cuando la cantidad disponible supere la mitad de la reclamación total (Villar, 2005).

Apliquemos ahora la regla del Talmud al caso del pueblo donde el ayuntamiento tiene que repartir un fondo de ayuda de $E = 1000$ frente a unas reclamaciones de $c = (600,300,200,100)$.

3.4.1. Caso 1: $E \leq C/2$

Si el presupuesto disponible es menor o igual a la mitad de la reclamación agregada, es decir, $E \leq \frac{C}{2}$, utilizamos la primera parte de la regla del Talmud. En nuestro ejemplo, $\frac{C}{2} = 600$, y dado que $E = 1000$, en este caso no se aplicaría. Sin embargo, a modo de ilustración, vamos a suponer que $E = 500$.

De esta forma, calcularíamos λ resolviendo:

$$\sum_{i \in M} \min\left(\lambda, \frac{c_i}{2}\right) = 500. \tag{15}$$

De esta ecuación[3], obtenemos que $\lambda = 200$.

[3] Para resolver la ecuación se ha utilizado el método computacional de la bisección. Ver 5.3.1.

Dividimos las reclamaciones entre dos:

$$(16) \qquad \frac{c_1}{2} = 300, \frac{c_2}{2} = 150, \frac{c_3}{2} = 100, \frac{c_4}{2} = 50.$$

Aplicando la regla del Talmud:

$$(17) \quad T_1(500, c) = \min(200,300) = 200,$$
$$T_2(500, c) = \min(200,150) = 150,$$
$$T_3(500, c) = \min(200,100) = 100,$$
$$T_4(500, c) = \min(200,50) = 50.$$

De esta forma, observamos que usando la regla del Talmud y para el caso de que $E \leq \frac{C}{2}$, los vecinos con mayores reclamaciones tienen unas pérdidas proporcionalmente mayores comparado con los vecinos con menores reclamaciones.

3.4.2. Caso 1: $E \geq C/2$

Volvemos a suponer ahora que $E = 1000$. Dado que el presupuesto disponible es mayor que la mitad de la reclamación agregada ($E = 1000 > 600$), utilizamos la segunda parte de la regla del Talmud. De esta forma calculamos μ resolviendo:

$$(18) \qquad \sum_{i \in M} \max\left(\frac{c_i}{2}, c_i - \mu\right) = 1000.$$

De donde obtenemos[4] que $\mu = 50$. Calculamos:

$$(19) \qquad \frac{c_1}{2} = 300, \frac{c_2}{2} = 150, \frac{c_3}{2} = 100, \frac{c_4}{2} = 50.$$

Aplicando la regla del Talmud:

$$(20)$$

$$(20) \quad T_1(500, c) = \max(600 - 50,300) = 550,$$
$$T_2(500, c) = \max(300 - 50,150) = 250,$$
$$T_3(500, c) = \max(200 - 50,100) = 150,$$

[4] Para resolver la ecuación se ha utilizado el método computacional de la bisección. Ver 5.3.1.

$$T_4(500, c) = \max(100 - 50{,}50) = 50.$$

De esta forma, observamos que usando la regla del Talmud y para el caso de que $E \geq \frac{C}{2}$, los vecinos con menores reclamaciones tienen unas pérdidas proporcionalmente mayores comparado con los vecinos con menores reclamaciones.

3.5. Regla del orden de llegada

La finalidad de esta regla es la de no beneficiar a aquellos que antes llegan, y se puede calcular en dos pasos. Para el primer paso, se calculan las asignaciones que percibirían los agentes en función de todos los posibles órdenes de llegada. En el segundo paso, se calcula la media de las asignaciones que les hubieran correspondido y se asigna dicha cantidad. La regla, por tanto, tiene que contemplar todos los posibles órdenes de llegada de los agentes para establecer qué cantidad les sería otorgada y, posteriormente, calcular la media (Espinel Febles, 2007).

Tomemos como ejemplo el caso del ayuntamiento con $E = 1000$ y $c = (600, 300, 200, 100)$ y realicemos el primer paso. En este caso, existen $4! = 24$ posibles órdenes de llegada. Si el agente 1 aparece primero, se le asignaría la totalidad del presupuesto disponible (600 euros o lo que esté disponible) y a los demás agentes el resto del presupuesto disponible según el orden de llegada. Si el agente 2 llega primero, se le asignaría su demanda completa (300 euros) y al resto de los agentes se les asignaría lo que quede del presupuesto según el orden de llegada, y así sucesivamente para todos los órdenes posibles. En la Tabla 1, se muestra cómo se aplica la regla en este ejemplo específico, para todas las combinaciones posibles.

En el segundo paso, se calcula la media de todas las 24 asignaciones para cada agente reclamador. De esta forma, en este ejemplo, el agente 1 recibiría 541,67 euros, el agente 2 recibiría 241,67 euros, el agente 3 recibiría 141,67 euros y el agente 4 recibiría 75 euros. De esta forma, se pretende no beneficiar a aquellos que antes llegan cuando se ha declarado la bancarrota. De esta forma, podemos caracterizar formalmente la regla de orden de llegada como aquella que para cada problema (E, c) y para cada $i = 1, 2, \dots, m$ propone la asignación:

$$OL(E, c) = \left(\frac{\sum_{k=1}^{m!} x_{1k}}{m!}, \frac{\sum_{k=1}^{m!} x_{2k}}{m!}, \dots, \frac{\sum_{k=1}^{m!} x_{mk}}{m!} \right). \quad (21)$$

Donde x_k es la asignación a cada agente en función del orden de llegada, para cada una de las permutaciones $k = 1{,}2, \dots, m!$

Orden de Llegada	vecino 1	vecino 2	vecino 3	vecino 4
1234	600	300	100	0
1243	600	300	0	100
1324	600	200	200	0
1342	600	100	200	100
1423	600	300	0	100
1432	600	100	200	100
2134	600	300	100	0
2143	600	300	0	100
2314	500	300	200	0
2341	400	300	200	100
2413	600	300	0	100
2431	400	300	200	100
3124	600	200	200	0
3142	600	100	200	100
3214	500	300	200	0
3241	400	300	200	100
3412	600	100	200	100
3421	400	300	200	100
4123	600	300	0	100
4132	600	100	200	100
4213	600	300	0	100
4231	400	300	200	100
4312	600	100	200	100
4321	400	300	200	100
Suma	13000	5800	3400	1800
Asignación final	541,67	241,67	141,67	75

Tabla 1. Reparto según la regla del orden de llegada para el ejemplo de las inundaciones.

De esta forma, la asignación final para cada orden de llegada será:

$$(22) \qquad OL_1(E,c) = \frac{13000}{24} = 541{,}67,$$

$$OL_2(E,c) = \frac{5800}{24} = 241{,}67,$$

$$OL_3(E,c) = \frac{3400}{24} = 141{,}67,$$

$$OL_4(E,c) = \frac{1800}{24} = 75.$$

De esta forma, observamos cómo la regla de orden de llegada permite repartir el presupuesto independientemente del orden de llegada de los agentes.

4. DISEÑO Y ELABORACIÓN DE UN SCRIPT EN PYTHON PARA APLICAR LAS REGLAS DE REPARTO

Con el fin de poder utilizar, aplicar y comprender mejor las reglas de reparto, se ha realizado para este trabajo el diseño y programación de un script en Python. El script permite aplicar las reglas a partir de unas reclamaciones y un presupuesto dados por el usuario. A continuación, se detallarán las herramientas utilizadas, se explicará cómo utilizarlo, y se describirá el funcionamiento interno del mismo.

4.1. Herramientas utilizadas

4.1.1. Lenguaje de programación: Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su sintaxis sencilla y legibilidad, lo que lo hace accesible tanto para programadores experimentados como para principiantes. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python soporta múltiples paradigmas de programación, incluidos los imperativos, orientados a objetos y funcionales, lo que lo convierte en una herramienta muy versátil (Python Software Foundation, 2024).

La resolución de problemas de bancarrota implica la implementación de varias reglas de reparto y el análisis de los resultados. Python es una herramienta ideal para esta tarea ya que tiene una buena capacidad para manejar datos, y una gran disponibilidad de bibliotecas de estadística y visualización. En programación, una biblioteca es un conjunto de herramientas que pueden ser utilizadas durante el desarrollo de software. A continuación, explicaremos las principales bibliotecas utilizadas.

4.1.2. Librerías de Python

Pandas, *Matplotlib* y *NumPy* son bibliotecas empleadas ampliamente en el campo del análisis de datos y la ciencia de datos. En este trabajo, son las que han sido utilizadas para poder procesar los datos de entrada y representar los datos de salida.

Pandas, creada por Wes McKinney (Mckinney, 2012), permite usar estructuras de datos como *DataFrame* y *Series*, facilitando la manipulación de datos tabulares, permitiendo así la lectura y procesado de datos de reclamaciones desde un archivo CSV. *Matplotlib*, desarrollada por John D. Hunter (Hunter, 2007), es utilizada para generar gráficos en 2D que permiten visualizar cómo varían las asignaciones según el presupuesto disponible y las

diferentes reglas de reparto. *NumPy*, creada por Travis Oliphant (Oliphant, 2006), es fundamental para la computación científica, ofreciendo soporte para matrices y arreglos multidimensionales, y es utilizada en el script para realizar cálculos numéricos,, como la resolución de ecuaciones necesarias para determinar los valores de λ y δ.

4.1.3. Visual Studio Code

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft, lanzado por primera vez en 2015. Es conocido por su rapidez, flexibilidad y extensibilidad, ofreciendo soporte para una amplia gama de lenguajes de programación a través de extensiones. VS Code proporciona características avanzadas como depuración integrada y un terminal embebido, lo que facilita el flujo de trabajo de desarrollo (Microsoft, 2024).

En este trabajo, se ha utilizado Visual Studio Code por su entorno de desarrollo robusto y personalizable, que permite una edición de código eficiente, depuración y gestión de proyectos en Python, lo que es esencial para implementar y probar los algoritmos de este trabajo y para generar visualizaciones claras y precisas.

4.2. Modo de empleo

A continuación se describen los pasos a seguir par poder utilizar el script. (puede descargarse en un repositorio público de Github a través del enlace: https://github.com/VictorHerrezuelo/bancarrota) . En las figuras se han utilizado los datos del ejemplo del ayuntamiento y la inundación.

4.2.1. Comprobar que se tiene Python instalado.

Para verificar si Python está instalado en el sistema, hay que abrir el símbolo del sistema (*Windows+R, después escribir cmd y pulsar intro*) y escribir `python --version`. Si Python no está instalado, es necesario descargarlo e instalarlo desde python.org (https://www.python.org/).

4.2.2. Comprobar que se tienen las bibliotecas instaladas

Para verificar si las bibliotecas necesarias están instaladas, hay que escribir los siguientes comandos en el símbolo del sistema:

```
pip show pandas
pip show matplotlib
pip show numpy
```

Si alguna biblioteca no está instalada, es necesario instalarla con los siguientes comandos:

```
pip install pandas
pip install matplotlib
pip install numpy
```

4.2.3. Descargar el script y adaptar la plantilla

Descargar el fichero `reglas_reparto.py` y la plantilla de datos `plantilla.csv`. Para hacerlo, simplemente hay que acceder al enlace, hacer clic en el botón verde “Code” y descargar en “Download ZIP”. La plantilla puede ser editada para introducir los datos del problema. Cuenta con una columna con los nombres de los agentes, y otra columna con las reclamaciones de cada uno, tal y como se muestra en la Figura 1.

	A	B
1	agente_nombre	reclamacion
2	1	600
3	2	300
4	3	200
5	4	100
6		

Figura 1. Plantilla en CSV rellenada con los datos del ejemplo. Elaboración propia.

4.2.4. Ejecutar el script

Abrir el símbolo del sistema, navegar hasta el directorio donde se guardaron los ficheros y ejecutar el script con:

```
python reglas_reparto.py
```

4.2.5. Seleccionar CSV

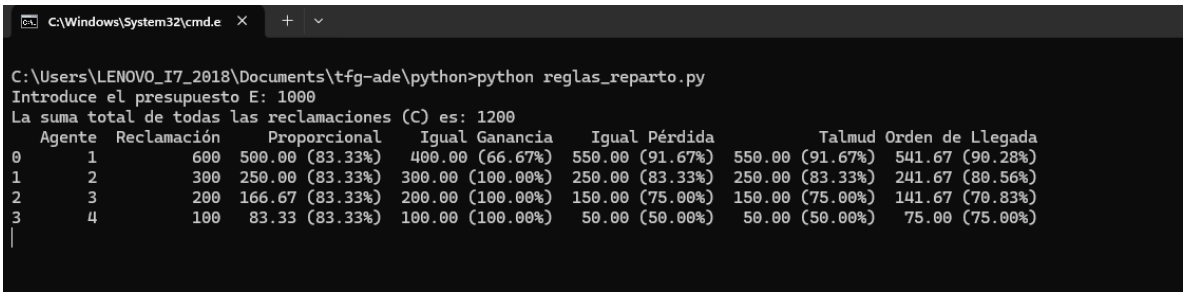
Al ejecutar el script, se abrirá un cuadro de diálogo para seleccionar el archivo CSV que contiene las reclamaciones de los agentes. Seleccionar el archivo adecuado y hacer clic en "Abrir".

4.2.6. Introducir el presupuesto

Después de seleccionar el archivo CSV, el script pedirá que se introduzca el presupuesto disponible. Ingresar el valor del presupuesto en la consola y presionar *Enter*.

4.2.7. Visualización de resultados. Guardar imagen en un fichero

El script generará gráficos que muestran cómo varían las asignaciones según el presupuesto disponible y las diferentes reglas de reparto. Para guardar la imagen de los resultados, hacer clic en el icono de guardar abajo a la izquierda.



```
C:\Windows\System32\cmd.e

C:\Users\LENOVO_I7_2018\Documents\tfg-ade\python>python reglas_reparto.py
Introduce el presupuesto E: 1000
La suma total de todas las reclamaciones (C) es: 1200
   Agente  Reclamación     Proporcional    Igual Ganancia    Igual Pérdida          Talmud Orden de Llegada
0       1          600  500.00 (83.33%)   400.00 (66.67%)  550.00 (91.67%)  550.00 (91.67%)  541.67 (90.28%)
1       2          300  250.00 (83.33%)  300.00 (100.00%)  250.00 (83.33%)  250.00 (83.33%)  241.67 (80.56%)
2       3          200  166.67 (83.33%)  200.00 (100.00%)  150.00 (75.00%)  150.00 (75.00%)  141.67 (70.83%)
3       4          100   83.33 (83.33%)  100.00 (100.00%)   50.00 (50.00%)   50.00 (50.00%)   75.00 (75.00%)
```

Figura 2. Ejecución en el símbolo del sistema del script en Python. Elaboración propia.

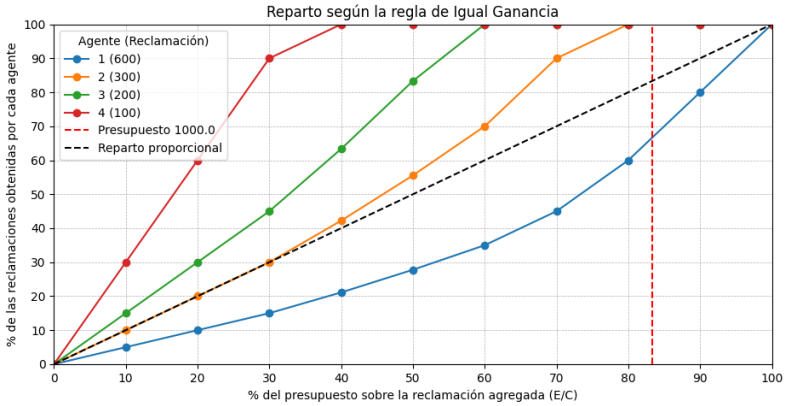


Figura 3. Regla de igual ganancia. Proporción de las reclamaciones obtenidas por cada agente vs barrido de 0 a 100% de E/C. Elaboración propia.

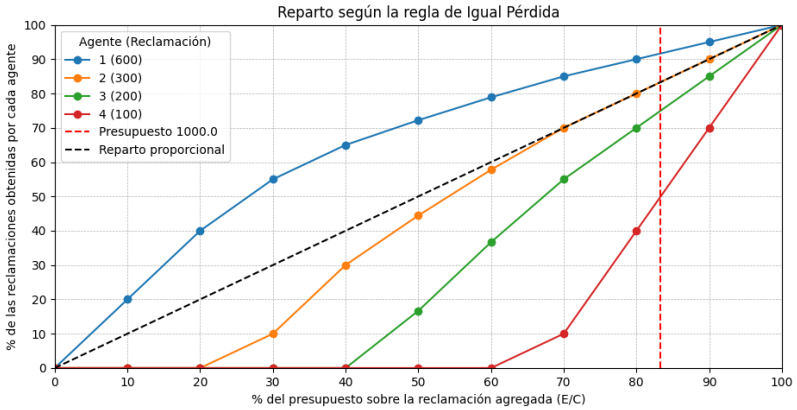


Figura 4. Regla de igual pérdida. Proporción de las reclamaciones obtenidas por cada agente vs barrido de 0 a 100% de E/C. Elaboración propia.

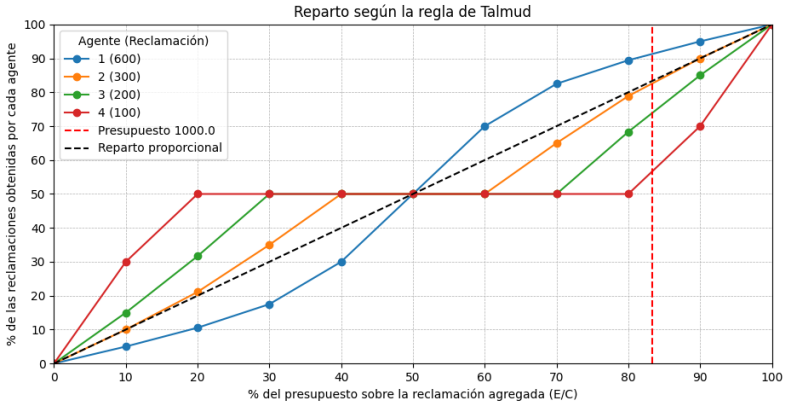


Figura 5. Regla del Talmud. Proporción de las reclamaciones obtenidas por cada agente vs barrido de 0 a 100% de E/C. Elaboración propia.

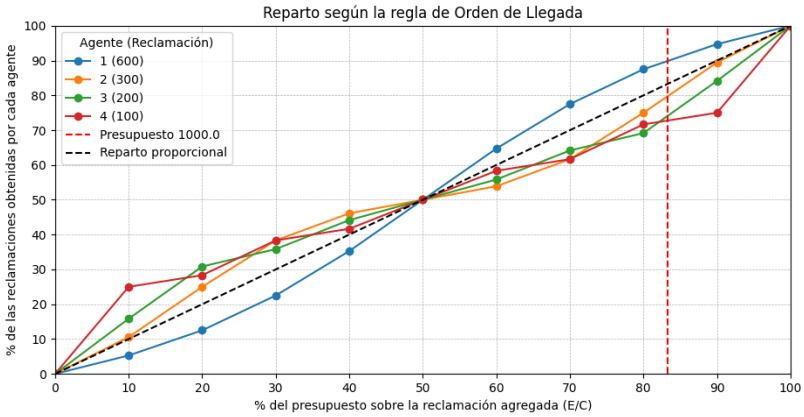


Figura 6. Regla del orden de llegada. Proporción de las reclamaciones obtenidas por cada agente vs barrido de 0 a 100% de E/C. Elaboración propia.

4.3. Estructura del código

El código se estructura en varias funciones principales que gestionan diferentes aspectos del problema de reparto en un escenario de bancarrota. Primero, la función `leer_datos` utiliza la biblioteca abre un cuadro de diálogo que permite al usuario seleccionar un archivo CSV con las reclamaciones de los agentes. Esta función lee el archivo y lo carga en un `DataFrame` para su posterior manipulación.
Las funciones de cálculo (`calcular_reparto_proporcional`, `calcular_reparto_igual_ganancia`, `calcular_reparto_igual_perdida`, `calcular_reparto_talmud` y `calcular_reparto_orden_llegada`) implementan diferentes reglas de reparto utilizando las bibliotecas `numpy`.

La representación de los resultados se realiza mediante la función `dibujar_graficas`, que genera gráficos con `matplotlib` para mostrar cómo varían las asignaciones según el presupuesto disponible en función de las diferentes reglas de reparto. Además, la función `imprimir_tabla` crea una tabla que muestra las reclamaciones de cada agente y las asignaciones resultantes para cada regla de reparto, junto con el porcentaje de la reclamación obtenida. Finalmente, la función `main` orquesta la ejecución del programa, llamando a las funciones de lectura de datos, entrada del presupuesto, impresión de la tabla y generación de gráficos.

4.3.1. Funciones para el cálculo de λ y μ

Para hallar λ y μ, es necesario resolver una serie de ecuaciones (ecuaciones 5, 9, 13 y 14) que por su complejidad se ha procedido a resolver por el método de la bisección (Alonzo Velázquez, 2011). El *método de la bisección* es un algoritmo de búsqueda de raíces que se utiliza para encontrar soluciones aproximadas a ecuaciones no lineales. Este método se basa en el principio de que una función continua que cambia de signo en un intervalo cerrado $[a, b]$ (en el código, `[low, high]`) debe tener al menos una raíz en ese intervalo. El proceso iterativo comienza dividiendo el intervalo $[a, b]$ a la mitad para encontrar el punto medio m (en nuestro código, `mid`). Luego, se evalúa la función en m para determinar en cuál de los subintervalos $[a, m]$ o $[m, b]$ la función cambia de signo.

El subintervalo donde se produce el cambio de signo se convierte en el nuevo intervalo de búsqueda, y el proceso se repite hasta que la longitud del intervalo sea menor que una tolerancia predefinida ϵ (en nuestro caso, `epsilon = 10e-6`), asegurando así una aproximación de la raíz con la precisión deseada.

Este algoritmo ha sido implementado en las funciones `calcular_lambda` y `calcular_mu`, como se puede ver en la Figura 7 y en la Figura 8. Estas funciones incorporan

un parámetro binario de entrada (`es_Talmud`) para determinar si queremos calcular λ y μ para la regla de igual ganancia, para la regla de igual ganancia, o para la regla del Talmud.

```python
def calcular_lambda(reclamaciones, E, es_talmud=False):
    # Función para calcular el valor de lambda
    # Para poder resolver la ecuación, se utiliza una búsqueda binaria
    if es_talmud:
        # En el caso de la regla de Talmud, las reclamaciones se dividen entre 2
        reclamaciones = reclamaciones / 2

    low, high = 0, max(reclamaciones) # Inicializamos los valores de low y high
    epsilon = 1e-6  # Tolerancia para la solución

    while high - low > epsilon:
        mid = (low + high) / 2
        suma_minimos = np.sum(np.minimum(mid, reclamaciones))
        if suma_minimos < E:
            low = mid
        else:
            high = mid

    return (low + high) / 2
```

Figura 7. Función `calcular_lambda` del script. Elaboración propia.

```python
def calcular_mu(reclamaciones, E, es_talmud=False):
    # Función para calcular el valor de mu
    # Para poder resolver la ecuación, se utiliza una búsqueda binaria
    low, high = 0, max(reclamaciones)
    epsilon = 1e-6  # Tolerancia para la solución

    while high - low > epsilon:
        mid = (low + high) / 2
        if es_talmud:
            suma_maximos = np.sum(np.maximum(reclamaciones / 2, reclamaciones - mid))
        else:
            suma_maximos = np.sum(np.maximum(0, reclamaciones - mid))

        if suma_maximos > E:
            low = mid
        else:
            high = mid

    return (low + high) / 2
```

Figura 8. Función `calcular_mu` del script. Elaboración propia.

4.3.2. Funciones para calcular las reglas de reparto

Como se puede ver en la Figura 8 y en la Figura 9, las funciones que calculan las reglas de reparto proporcional, de igual ganancia, de igual pérdida y del Talmud meramente implementan las ecuaciones (ver sección 3).

Sin embargo, más atención merece la función que implementa la regla de orden de llegada. Primero, se generan todas las permutaciones posibles del orden de llegada de los agentes

usando la función `itertools.permutations` de la biblioteca `numpy`, lo cual proporciona una lista de todas las secuencias en las que los agentes pueden llegar. Luego, se inicializa el vector auxiliar `distribucion` con ceros, que almacenará más adelante la suma acumulada de los repartos para todas las permutaciones. Para cada permutación, se hace una copia del presupuesto E y se inicializa otra variable auxiliar `temp_distribucion` con ceros para registrar el reparto específico de esa permutación.

Dentro del bucle que itera sobre cada permutación, se distribuye el presupuesto entre los agentes según el orden de llegada definido por la permutación. Para cada agente en la permutación, se asigna el mínimo entre la reclamación del agente y el presupuesto restante (variable auxiliar `E_temp`). Esta asignación se almacena en `temp_distribucion` y se reduce el presupuesto restante (`E_temp`). Si el presupuesto se agota (`E_temp <= 0`), se rompe el bucle interno. Después de calcular el reparto para la permutación actual, `temp_distribucion` se añade a `distribucion`. Una vez que se han considerado todas las permutaciones, se obtiene la media de las distribuciones dividiendo `distribucion` por el número total de permutaciones. Así, la función devuelve el reparto promedio para cada agente, considerando todas las posibles secuencias de llegada.

```python
def calcular_reparto_proporcional(reclamaciones, E):
    # Función para calcular el reparto proporcional
    C = reclamaciones.sum()
    return (E / C) * reclamaciones

def calcular_reparto_igual_ganancia(reclamaciones, E):
    # Función para calcular el reparto con igual ganancia
    lambda_value = calcular_lambda(reclamaciones, E)
    return np.minimum(reclamaciones, lambda_value)

def calcular_reparto_igual_perdida(reclamaciones, E):
    #  Función para calcular el reparto con igual pérdida
    mu_value = calcular_mu(reclamaciones, E)
    return np.maximum(reclamaciones - mu_value, 0)

def calcular_reparto_talmud(reclamaciones, E):
    # Función para calcular el reparto con la regla de Talmud
    C = reclamaciones.sum()
    half_c = C / 2
    if E <= half_c:
        lambda_value = calcular_lambda(reclamaciones, E, es_talmud=True)
        return np.minimum(reclamaciones / 2, lambda_value)
    else:
        mu = calcular_mu(reclamaciones, E, es_talmud=True)
        return np.maximum(reclamaciones / 2, reclamaciones - mu)
```

Figura 9. Funciones del script que calculan las asignaciones para las diferentes reglas de reparto. Elaboración propia.

```python
def calcular_reparto_orden_llegada(reclamaciones, E):
    # Función para calcular el reparto según la regla del orden de llegada
    n = len(reclamaciones)
    permutaciones = list(itertools.permutations(range(n))) # Todas las permutaciones posibles
    distribucion = np.zeros(n)
    for perm in permutaciones:
        # Calculamos el reparto para cada permutación
        E_temp = E
        temp_distribucion = np.zeros(n)
        for agente in perm:
            # Asignamos el mínimo entre la reclamación del agente y el presupuesto restante
            asignado = min(reclamaciones[agente], E_temp)
            temp_distribucion[agente] = asignado
            E_temp -= asignado
            if E_temp <= 0:
                break
        distribucion += temp_distribucion
    return distribucion / len(permutaciones) # Devolvemos la media de los repartos
```

Figura 10. Función del script que calcula las asignaciones para la regla del orden de llegada. Elaboración propia.

4.3.3. Funciones para el dibujado de las gráficas

La función `dibujar_graficas` genera gráficos que muestran cómo varían las asignaciones según el presupuesto disponible y las diferentes reglas de reparto. Primero, se configuran las `subplots` (subgráficas) para poder encuadrar cuatro gráficas en una misma figura.

Se calcula el porcentaje del presupuesto E respecto a la reclamación agregada C para representar visualmente este valor en los gráficos. Para cada regla de reparto, y para cada agente, se calcula el porcentaje de la reclamación obtenida para distintos porcentajes del presupuesto total. Además, se añade una línea roja punteada para marcar el presupuesto real E introducido por el usuario. Se dibuja también una línea negra discontinua para el reparto proporcional ideal.

La función `imprimir_tabla` crea una tabla que presenta las reclamaciones de cada agente y las asignaciones resultantes según diferentes reglas de reparto para un presupuesto específico E.

```python
def dibujar_graficas(resultados, reclamaciones, C, E):
    # Función para dibujar las gráficas con los resultados
    fig, axs = plt.subplots(2, 2, figsize=(15, 10))
    reglas = ['Igual Ganancia', 'Igual Pérdida', 'Talmud', 'Orden de Llegada']
    porcentajes = np.linspace(0, 1, 11)  # Coordenadas x (0, 0.1, ..., 1)
    E_porcentaje = (E / C) * 100

    # Dibujamos las gráficas
    for i, regla in enumerate(reglas):
        ax = axs[i // 2, i % 2]
        for j, agente in enumerate(resultados['Agente']):
            porcentaje_obtenido = [0.0]  # Comenzamos con (0,0)
            for p in porcentajes[1:-1]:  # Excluimos el primer y último punto
                E_temp = p * C
                repartido = calcular_reparto(reclamaciones, E_temp, regla)
                porcentaje_obtenido.append((repartido[j] / reclamaciones[j]) * 100 if reclamaciones[j] != 0 else 0)
            porcentaje_obtenido.append(100.0)  # Terminamos con (1,1)

            ax.plot(porcentajes * 100, porcentaje_obtenido, label=f'{agente} ({reclamaciones.iloc[j]})', marker='o')

        ax.axvline(x=E_porcentaje, color='r', linestyle='--', label=f'Presupuesto {E}')
        ax.plot([0, 100], [0, 100], 'k--', label='Reparto proporcional')
        ax.set_title(f'Reparto según la regla de {regla}')
        ax.set_xlabel('% del presupuesto sobre la reclamación agregada (E/C)')
        ax.set_ylabel('% de las reclamaciones obtenidas por cada ')
        ax.set_xlim([0, 100])
        ax.set_ylim([0, 100])
        ax.xaxis.set_major_locator(plt.MultipleLocator(10))
        ax.yaxis.set_major_locator(plt.MultipleLocator(10))
        ax.grid(True, which='both', axis='both', linestyle='--', linewidth=0.5)
        ax.legend(title="Agente (Reclamación)")

    plt.tight_layout()
    plt.show()
```

Figura 11. Función que dibuja las gráficas de barrido. Elaboración propia.

```python
def imprimir_tabla(resultados, reclamaciones, E):
    # Función para imprimir la tabla con los resultados
    reglas = ['Proporcional', 'Igual Ganancia', 'Igual Pérdida', 'Talmud', 'Orden de Llegada']
    tabla = pd.DataFrame({'Agente': resultados['Agente'], 'Reclamación': reclamaciones})

    for regla in reglas:
        repartos = calcular_reparto(reclamaciones, E, regla)
        porcentajes = (repartos / reclamaciones) * 100
        tabla[f'{regla}'] = [f'{reparto:.2f} ({porcentaje:.2f}%)' for reparto, porcentaje in zip(repartos, porcentajes)]

    print(tabla)
```

Figura 12. Función que imprime por pantalla la tabla de asignaciones para el presupuesto y reclamaciones dadas. Elaboración propia.

5. APLICACIÓN A UN CASO REAL: MARCO FINANCIERO PLURIANUAL Y FONDOS NEXTGENERATIONEU DE LA UNIÓN EUROPEA

Tras la pandemia de la Covid-19, la Unión Europea anunció varios programas de recuperación basados en la financiación de diferentes áreas de desarrollo. Dos de estos programas son el **Marco Financiero Plurianual (MFP)** y los fondos **NextGenerationEU (NextGenEU)**. La combinación de ambos constituye el mayor paquete de estímulo financiero jamás emitido en Europa, totalizando más de 2 billones de euros a precios corrientes.

El Marco Financiero Plurianual (MFP) es un presupuesto a largo plazo de la UE (2021-2027) destinado a apoyar la recuperación económica, la cohesión territorial, la transición ecológica y digital, la seguridad y defensa, y la gestión de fronteras, entre otros objetivos. La financiación se distribuye a través de varias rúbricas, cada una destinada a diferentes áreas prioritarias, asegurando que los recursos se utilicen de manera eficiente y equitativa para enfrentar los desafíos más urgentes de la Unión Europea. Por otra parte, NextGenerationEU (NextGenEU) es un instrumento temporal de recuperación financiado por la Unión Europea con más de 800 000 millones de euros, diseñado específicamente para reparar los daños económicos y sociales inmediatos causados por la pandemia de COVID-19. El principal componente de NextGenEU es el Mecanismo de Recuperación y Resiliencia (MRR), que proporciona subvenciones y préstamos a los Estados miembros de la UE para apoyar reformas e inversiones (Dirección General de Comunicación de la Comisión Europea, 2021).

A continuación, utilizaremos el script de Python diseñado en el apartado anterior para simular un reparto de los fondos de NextGenEU si el presupuesto se redujera en un 25%. Este ejercicio permitirá visualizar cómo las distintas reglas de reparto afectarían la distribución de los fondos entre los Estados miembros en un escenario de reducción presupuestaria.

La asignación de las partidas de los dos instrumentos se puede ver en la siguiente tabla:

Número - Partida	MFP	NextGenerationEU	Total
1. Mercado único, innovación y economía digital	€ 149.500.000,00	€ 11.500.000,00	€ 161.000.000,00
2. Cohesión, resiliencia y valores	€ 426.700.000,00	€ 776.500.000,00	€ 1.203.200.000,00
3. Recursos naturales y medio ambiente	€ 401.000.000,00	€ 18.900.000,00	€ 419.900.000,00
4. Migración y gestión de las fronteras	€ 25.700.000,00	€ -	€ 25.700.000,00
5. Seguridad y defensa	€ 14.900.000,00	€ -	€ 14.900.000,00
6. Vecindad y resto del mundo	€ 110.600.000,00	€ -	€ 110.600.000,00
7. Administración pública europea	€ 82.500.000,00	€ -	€ 82.500.000,00
Total	€ 1.210.900.000,00	€ 806.900.000,00	€ 2.017.800.000,00

Tabla 2. Asignación de los fondos europeos a las diferentes partidas. Fuente: (Dirección General de Comunicación de la Comisión Europea, 2021)

Vemos de esta forma que la asignación total agregada de los dos programas (MFP y NextGenEU) suma un total de 2.017,8 millones de euros. Ahora bien, si asumiéramos una reducción de, por ejemplo, el 25% del presupuesto total, nos podemos preguntar cuál va a

ser el efecto sobre las reclamaciones si empleáramos cada una de las reglas de bancarrota. Para poder averiguarlo, utilizaremos el script diseñado en la sección anterior.

Para ello, primero rellenamos el CSV con los datos de las asignaciones de fondos a cada partida. A continuación, iniciamos el script e introducimos el presupuesto, que resulta de multiplicar la asignación total por 75% (esto es, 1.513.350.000 euros). Estos dos primeros pasos se muestran en la Figura 13.

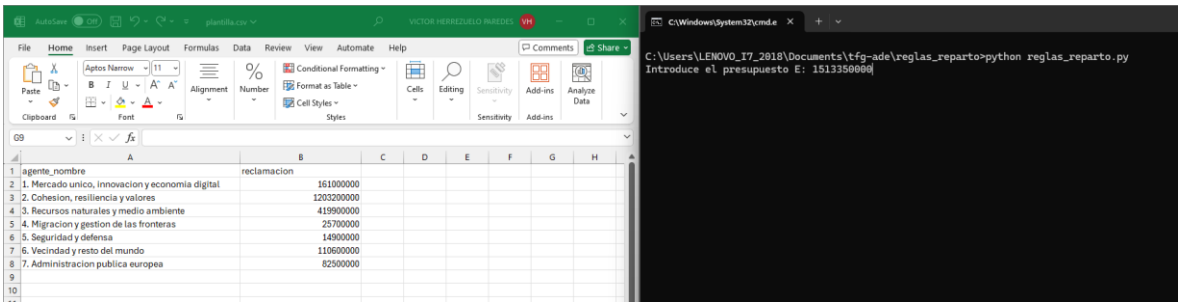


Figura 13. Uso del script para introducir los datos de los fondos europeos.

De esta forma, obtenemos los resultados para el reparto que se pueden ver en la Figura 14. En la tabla 3, se representa de forma más clara dichos resultados.

Los resultados de aplicar las diferentes reglas de reparto a los fondos europeos muestran cómo cada método distribuye los recursos de manera distinta. La regla proporcional distribuye el presupuesto disponible de manera uniforme, asignando el 75% de las reclamaciones a todas las partidas. Por otro lado, la regla de igual ganancia permite que la mayor parte de las partidas reciban el 100% de sus reclamaciones. En cambio, en el caso de la regla de igual pérdida, varias partidas se quedarían con el 0% de sus reclamaciones. Esto se debe a la gran diferencia en las cuantías de las diferentes partidas.

Observamos también en este ejemplo cómo la regla del Talmud trata de equilibrar las dos reglas anteriores, pues 5 de las 7 partidas obtendrían el 50% de sus reclamaciones. Finalmente, la regla del orden de llegada distribuye los fondos asignando valores cercanos al 63% a varios de las partidas.

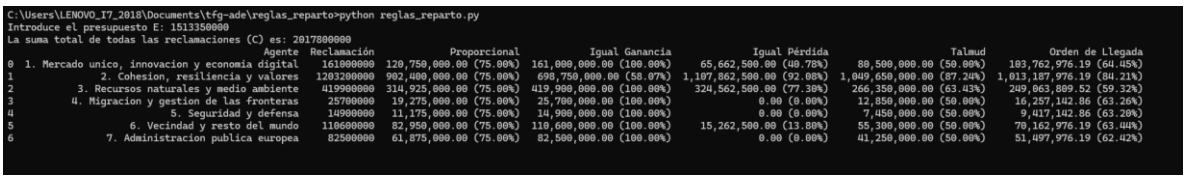


Figura 14. Reparto de fondos europeos con una reducción del 25% del presupuesto usando las cinco reglas de reparto.

Partida N#	Reclamación inicial	Proporcional		Igual Ganancia		Igual Pérdida	
1	161.000.000,00 €	120.750.000,00 €	75,00%	161.000.000,00 €	100,00%	65.662.500,00 €	40,78%
2	1.203.200.000,00 €	902.400.000,00 €	75,00%	698.750.000,00 €	58,07%	1.107.862.500,00 €	92,08%
3	419.900.000,00 €	314.925.000,00 €	75,00%	419.900.000,00 €	100,00%	324.562.500,00 €	77,30%
4	25.700.000,00 €	19.275.000,00 €	75,00%	25.700.000,00 €	100,00%	- €	0,00%
5	14.900.000,00 €	11.175.000,00 €	75,00%	14.900.000,00 €	100,00%	- €	0,00%
6	110.600.000,00 €	82.950.000,00 €	75,00%	110.600.000,00 €	100,00%	15.262.500,00 €	13,80%
7	82.500.000,00 €	61.875.000,00 €	75,00%	82.500.000,00 €	100,00%	- €	0,00%
Total	2.017.800.000,00 €	1.513.350.000,00 €	75,00%	1.513.350.000,00 €	75,00%	1.513.350.000,00 €	75,00%

Partida N#	Reclamación inicial	Talmud		Orden de Llegada	
1	161.000.000,00 €	80.500.000,00 €	50,00%	103.762.976,19 €	64,45%
2	1.203.200.000,00 €	1.049.650.000,00 €	87,24%	1.013.187.976,19 €	84,21%
3	419.900.000,00 €	266.350.000,00 €	63,43%	249.063.809,52 €	59,32%
4	25.700.000,00 €	12.850.000,00 €	50,00%	16.257.142,86 €	63,26%
5	14.900.000,00 €	7.450.000,00 €	50,00%	9.417.142,86 €	63,20%
6	110.600.000,00 €	55.300.000,00 €	50,00%	70.162.976,19 €	63,44%
7	82.500.000,00 €	41.250.000,00 €	50,00%	51.497.976,19 €	62,42%
Total	2.017.800.000,00 €	1.513.350.000,00 €	75,00%	1.513.350.000,00 €	75,00%

Tabla 3. Aplicación de las cinco reglas de reparto a una hipotética reducción del 25% en la asignación de fondos europeos.

Veamos finalmente cómo se comportaría la asignación de fondos en función del presupuesto, utilizando para ello el mismo script en Python. Se pueden ver los resultados en las Figuras 15, 16, 17, y 18.

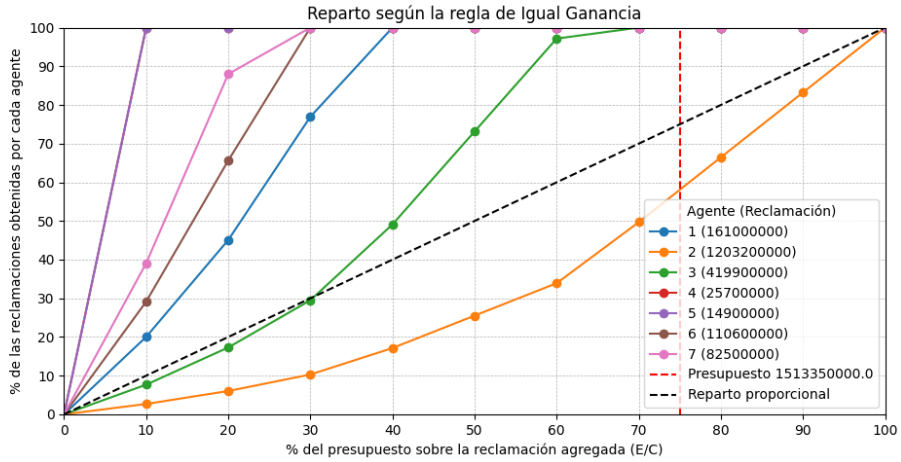


Figura 15. Regla de igual ganancia. Proporción de las reclamaciones obtenidas por cada partida de presupuesto vs barrido de 0 a 100% de E/C. Elaboración propia.

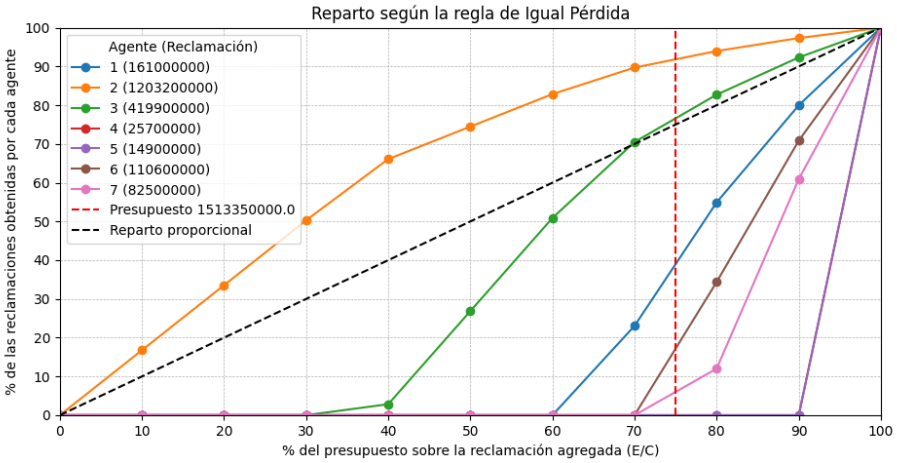


Figura 16. Regla de igual pérdida. Proporción de las reclamaciones obtenidas por cada partida de presupuesto vs barrido de 0 a 100% de E/C. Elaboración propia.

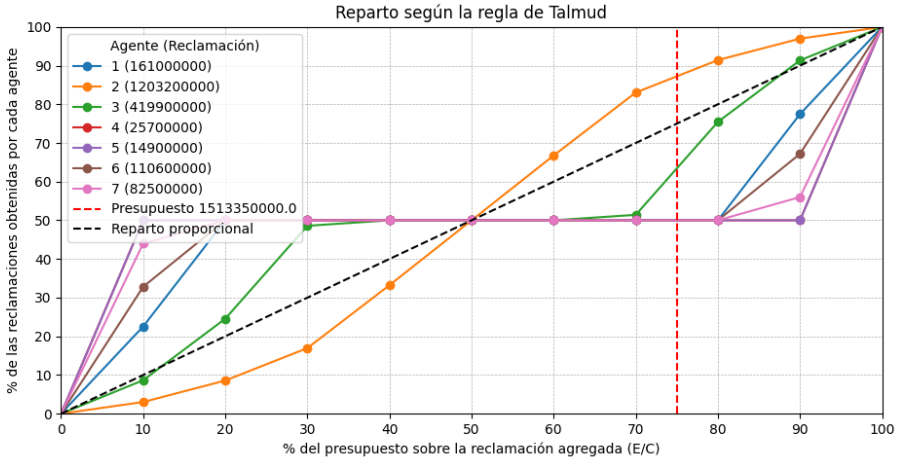


Figura 17. Regla del Talmud. Proporción de las reclamaciones obtenidas por cada partida de presupuesto vs barrido de 0 a 100% de E/C. Elaboración propia.

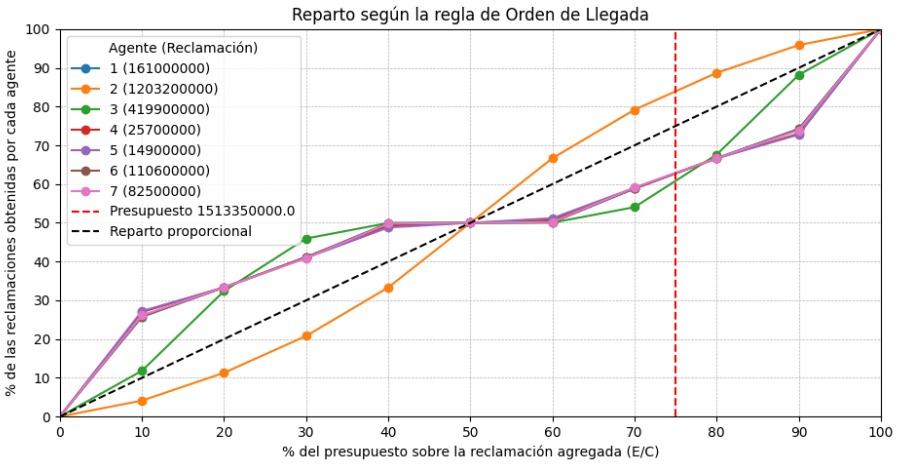


Figura 18. Regla del Orden de Llegada. Proporción de las reclamaciones obtenidas por cada partida de presupuesto vs barrido de 0 a 100% de E/C. Elaboración propia.

6. CONCLUSIONES Y LÍNEAS FUTURAS

En este trabajo, hemos explorado diversas reglas de reparto aplicables al problema de bancarrota, proporcionando tanto una explicación teórica como implementaciones prácticas en Python. A través de ejemplos y visualizaciones, se ha demostrado cómo las diferentes reglas de reparto afectan la distribución de recursos limitados entre los agentes. El uso de Python y bibliotecas como `pandas`, `numpy`, y `matplotlib` ha permitido la automatización de cálculos complejos y la generación de gráficos que facilitan la comprensión de los resultados. Como hemos podido comprobar, el uso de soluciones computacionales puede ayudar a resolver ecuaciones de una forma rápida y sencilla, como las empleadas para hallar los valores de λ y μ en el ejemplo.
Más allá de ese trabajo, hay tres líneas futuras de investigación que pueden ser dignas de exploración.

En primer lugar, la aplicación informática puede ser utilizada y perfeccionada con el fin de usarla para resolver problemas de reparto más complejos y realizar análisis más profundos.

En segundo lugar, la aplicación puede ser modificada para servir como aplicación interactiva y educativa que enseñe al usuario el funcionamiento y los resultados de las diferentes reglas de reparto. Para lograr esto, por ejemplo, se podría mostrar por pantalla la explicación teórica de las reglas de reparto. Una vez introducidos los datos, la aplicación podría ir explicando paso a paso la obtención de los repartos correspondientes a las diferentes reglas.

Por último, puede tener cabida realizar un análisis de sensibilidad más exhaustivo para entender cómo variaciones en las reclamaciones y el presupuesto afectan las asignaciones finales. De esta forma, podríamos analizar más en profundidad las propiedades de las mismas, y tener más información a la hora de decidir qué regla es mejor aplicar en función de las circunstancias.

7. BIBLIOGRAFÍA

Aumann, R. J., & Maschler, M. (1985). Game theoretic analysis of a bankruptcy. *Journal of Economic Theory, 26*, 195-213.

Espinel Febles, M. C. (2007). El reparto de lo escaso. *UNIÓN. Revista Iberoamericana de Educación Matemática*, 95-108.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 90-95.

Martínez Panero, M., & Meneses Poncio, L. C. (2011). Propuesta para seleccionar una solución en un problema de bancarrota. *Departamento de Economía Aplicada. Grupo de Investigación PRESAD. Universidad de Valladolid*.

Mckinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* O'Reilly Media, Inc.

Microsoft. (2024). *Documentatio for Visual Studio.* Retrieved from Visual Studio Code: https://code.visualstudio.com/docs

Oliphant, T. (2006). *Guide to NumPy.* Retrieved from Massachusets Institute of Technology: https://web.mit.edu/dvp/Public/numpybook.pdf

Python Software Foundation. (2024). *About Python*. Obtenido de python.org: https://www.python.org/about/

Villar, A. (2005). Cómo repartir cuando no hay bastante. *Lecturas de Economía - Universidad de Antioquía*, 9-34.

Villar, A., & Herrero, C. (2000). The three musketeers: four classical solutions to bankruptcy problems. *Instituto Valenciano de Investigaciones Económicas, s.a.*

Universidad de Valladolid

VISTO BUENO DEL TUTOR/ES PARA LA PRESENTACIÓN DEL TRABAJO FIN DE GRADO

Según lo dispuesto en el Real Decreto 1393/2007, de 29 de octubre, por el que se establece la ordenación de las enseñanzas universitarias oficiales, y como profesor tutor del trabajo de fin de estudios en el título de

Grado en ITTADE

de la (VA) Facultad de Ciencias Económicas y Empresariales

de la Universidad de Valladolid, D. Carlos Rodríguez Palmero

y

DECLARA/N que el/la estudiante D. Víctor Herrezuelo Paredes

ha realizado bajo su tutela el trabajo titulado:

CÁLCULO Y VISUALIZACIÓN DE CINCO REGLAS DE REPARTO PARA EL PROBLEMA DE LA BANCARROTA MEDIANTE PYTHON

Breve informe del tutor/es

Considera/n que el TFG anteriormente mencionado cumple los requisitos establecidos y AUTORIZA/N su presentación para la defensa ante la Comisión evaluadora correspondiente.

En Valladolid , a fecha de firma electrónica

Fdo.: El tutor/es