Linear Algebra Application in Games Theory: Finding Nash Equilibria in Bimatrix Games

Muhammad Adha Ridwan 13523098
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523098@std.stei.itb.ac.id

***Abstract*—Linear algebra provides tools for analyzing strategic interactions in game theory, especially for finding Nash equilibria for bimatrix games. This paper explores how fundamental concepts in linear algebra such as matrix operations and linear systems can be utilized to determine equilibrium points in two-player non-cooperative games. Basic matrix operations and a system of linear equations emerge as key techniques in analyzing these strategic interactions. Through numerical examples, this paper illustrate how these mathematical tools can help to identify Nash equilibria in games with both pure and mixed strategies, providing a straightforward approach to solving strategic decision problems.**

***Keywords*—Bimatrix games, Linear algebra, Nash equilibria.**

I. Introduction

Game theory has emerged as a crucial framework for analyzing strategic decision-making across various fields, ranging from economics to computer sciences. The principle, the analysis of games often relies on mathematical tools, using linear algebra concepts as the key techniques in solving two-player games. This intersection of game theory and linear algebra provides solutions to complex strategic problems.

In two-player games, often represented as bimatrix games, each player must make decisions while taking into account the potential strategies of their opponent. The concept of Nash equilibria, introduced by John Nash in 1950, provides a solution concept where no player can unilaterally enchance their payoff by altering their strategy. Identifying these equilibria necessitates a rigorous and systematic methematical approach

Linear algebra provides tools for this analysis. Payoff matrices, which represent the outcomes of each player's strategic choices, can be manipulated using basic matrix operations to identify pure strategy equilibria. When pure strategy equilibria do not exist, mixed strategies- where players randomize their choices according to probability distributions be found using systems of linear equations.

This paper explore how fundamental concepts from linear algebra can be applied to find Nash equilibria in bimatrix games. Starting with the mathematical foundations of game representation using matrices, followed by methods for identifying pure strategy equilibria. We then demonstrate how the system of linear equations can determine mixed strategy equilibria. Throughout the paper, we illustrate these concepts with practical examples of increasing complexity, from 2x2 games to 3x3 games.

II. Theoretical foundation

A. Game Theory

Game theory is the study of a mathematical model of strategic interactions between rational decision-makers, where the outcome for each participant depends not only on their actions but also on the actions of others. It is a tools that help to analyze how individuals or groups make decisions in a competitive and cooperative environment, where their choices are interdependent.

Modern game theory began with the idea of mixed-strategy equilibria in two-person zero-sum games and its proof by John von Neumann in 1928. Later on in 1944 John von Neumann and Oskar Morgenstern published Theory of Games and Economic Behavior expanded game theory to include cooperative and non-zero-sum games. In the 1950s John Nash developed the concept of Nash Equilibrium, showing that equilibria exist in non-zero-sum games.

In general, game theory can be defined as the mathematical study of optimal strategies that rational players adopt in interactive environments, where the decisions of one player influence the outcomes of the other.

Key elements in game theory are:

1. Players
   In game theory, players are the decision-makers that are involved in the strategic situation. These players may be individuals, groups, organizations, or even entire countries. Each player's goal is to maximize their own payoff or outcome, often while considering the strategies and actions of others.
2. Strategies
   A strategy is a plan of actions or moves that a player follows throughout the game. In some

cases, a player may choose a pure strategy, where they select a specific action. In other cases, a mixed strategy may be used, where a player chooses among possible actions according to probability distribution on moves.

3. Payoffs
   Payoffs represent the rewards or outcomes that players receive as a result of the actions they take during the game. The payoff can depend not only on a player's own strategy but also on the strategies chosen by other players. These payoffs are typically presented in the form of payoff matrices, which help to understand and visualize better the possible outcomes.
4. Game
   Games can be categorized based on their structure:
   a. Zero-sum games
      In zero-sum games, the total payoffs remain constant, meaning one player's winnings are another player's losses.
   b. Non-zero-sum games
      In contrast to zero-sum games, these games allow for the possibility that all players can benefit or suffer together, depending on the strategy they choose.
   c. Cooperative games
      Players can form some kind of alliance or coalition to achieve mutually beneficial rewards or outcomes.
   d. Non-cooperative games
      Players must act independently, making decisions without considering agreement.
5. Nash Equilibrium
   Nash equilibrium refers to a situation in which no player can improve their payoff by unilaterally changing their strategy, assuming that all other players' strategies remain fixed and no player has an incentive to deviate from their chosen strategy.

B. Nash Equilibrium

Nash equilibrium, introduced by Nash in 1950, constitutes a fundamental solution concept in non-cooperative game theory. Given a strategic-form game G = (N,$\{S_i\}$i∈N,$\{U_i\}$i∈N), where N denotes the sets of players, S_i denotes the strategy space of player I, and U_i represents player i's utility function, a strategy profile s* = (s_1*,s_2*,.., s_n*) constitutes a Nash equilibrium if for all player i∈N and all alternative strategies $s_i \in S_i$:

$$U_i\left(s_i^*, s_{-i}^*\right) \geqslant U_i\left(s_i, s_{-i}^*\right)$$

This mathematical formula captures the important stability property of Nash equilibrium, ensuring that no player can benefit from deviating from their equilibrium strategy while other players keep their strategies, based on the types there are a few types of Nash equilibrium:

a. Pure Strategy Nash Equilibrium
   Pure strategy Nash equilibrium occurs when players choose deterministic strategies from their own strategy sets. In this case, each player selects a single action with certainty rather than randomizing among multiple decisions. A pure strategy Nash equilibrium exists when there is a strategy profile where each player's pure strategy represents the best response to another player's pure strategies.
   We can't guarantee that there will always exist a pure strategy Nash equilibrium for every game. For example, in the matching pennies game, where two players simultaneously choose either head or tails, with one player winning if the choices match and the other winning if they differ, no pure strategy Nash equilibrium exists because any deterministic choice by one player can always be exploited by the other.
b. Mixed Strategy Nash Equilibrium
   Mixed strategy Nash equilibrium extends the equilibrium concept by allowing players to randomize over their pure strategies according to probability distributions. A mixed strategy σ_i for player i assigns the probabilities to each pure strategy in S_i. The expected utility for player i given strategy profile $\sigma = (\sigma_1, \sigma_2 .., \sigma_n)$ is calculated with the following formula:
   $E[U_i(\sigma)] = \Sigma s \in S\ [U_i(s) \times \Pi j \in N\ \sigma_j(s_j)]$
   Nash's existence theorem guarantees that every finite game has at least one mixed strategy Nash equilibrium. This ensures that strategic conditions modeled as games will always have a stable solution either pure or mixed.

C. Matrix

A matrix is a rectangular array of numbers arranged in rows and columns. For any elements A_{ij} where i = 1,2,3,4…,m and j = 1,2,3,4..,n is a matrix element positioned at row-*i* and column-*j* and i and j are the matrix indexes.

The order or dimension of the matrix is the product of the number of rows and columns denoted as m×n.

1. Matrix Type
   Matrix has many types, to simplify we will only discuss a few of its types.
   - Square Matrix
     A square matrix is a matrix where the number of rows and columns are equal, or denoted as n×n.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

   - Triangular Matrix
     A triangular matrix has two types, an upper triangular matrix and a lower triangular matrix. Upper triangular is a square matrix that has all elements under the main diagonal line all zero. Meanwhile, a lower triangular matrix is the opposite, all the elements above the main diagonal line are zero.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix}$$

- Transpose Matrix
  Transpose matrix is obtained by converting rows to columns and *vice versa.* Denoted as A^T.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

2. Matrix Operations
   Matrix operations are used to manipulate the elements of matrix, this enables us to find solutions of linear systems and transformations.
   - Scalar Multiplications
     Multiplicating a matrix with a scalar will result in a matrix of the same dimensions but with each of its elements multiplied by the scalar value.

$$k \times A = k \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ka & kb \\ kc & kd \end{bmatrix}$$

   - Matrix Multiplications
     Multiplicating a matrix with another matrix is a more complex operation than with a scalar.
     For example, matrix $A_{m\times n}$ multiplied by matrix $B_{x\times y}$ this operation could only if:
     - $A\times B$ is operatable only if n=x and the result matrix dimension is $m \times y$.
     - $B\times A$ is operatable only if y=m and the result matrix dimension is $x \times n$.
     The element of the first row and the second column is a sum of product elements from row 1 of matrix A with elements from column 2 of matrix B.
     For example:
     We have matrices A and B such as

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \qquad B = \begin{bmatrix} p & s \\ q & t \\ r & u \end{bmatrix}$$

     The product result matrix is

$$A \times B = \begin{bmatrix} ap + bq + cr & as + bt + cu \\ dp + eq + fr & ds + et + fu \end{bmatrix}$$

3. Elementary Row Operations
   Elementary row operations or often shortened ERO an arithmetic operations that are applied to each element in a row.
   Elementary row operations include:
   - Row Exchange ($R_i \leftrightarrow R_j$)
   - Scalar Multiplication (cR_i, $c \neq 0$)
   - Row Addition

The term used in ERO:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- Element at A_{11} (1) is called leading one
- Element at A_{22} (5) is called the first non-zero element
- The first, second, and third rows are called non-zero rows because the first element in each row is non-zero.
- The fourth row is called the zero row

The result of elementary row operations is a matrix in the form of a row echelon with these characteristics:
- For every non-zero row, the first element is always 1.
- For the next non-zero row, the first element is placed further right than the previous row.
- All zero rows should be placed in the bottom row.

Gauss-Jordan method of elementary row operations results in a matrix in the form of a reduced echelon row with all elements except the leading one is all zero. For example :

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 2 & -1 \\ 2 & 1 & -2 \end{bmatrix}$$

To find the row echelon matrix we will use elementary row operations:

$$b_2 - 3b_1 \sim \begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & -1 \\ 2 & 1 & -2 \end{bmatrix}$$

$$b_2/4 \sim \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & -\frac{1}{4} \\ 2 & 1 & -2 \end{bmatrix}$$

$$b_3 - 2b_1 \sim \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & -\frac{1}{4} \\ 0 & -3 & -2 \end{bmatrix}$$

$$b_3 + 3b_2 \sim \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & -\frac{1}{4} \\ 0 & 0 & -\frac{11}{4} \end{bmatrix}$$

$$b_3 / -\frac{11}{4} \sim \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & -\frac{1}{4} \\ 0 & 0 & 1 \end{bmatrix}$$

The resulting matrix is:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & -\frac{1}{4} \\ 0 & 0 & 1 \end{bmatrix}$$

4. Matrix Inverse

A square matrix A has an inverse only if there exists square matrix B such that :

$$A \times B = B \times A = I$$

With I is an identity matrix and B is the inverse of A or denoted as A^{-1} We can find an inverse of a matrix using elementary row operations using :

$$(A|B) \sim (I|A^{-1})$$

The steps are following:
- Use elementary row operations on the left side matrix A simultaneously with identity matrix on the right side.
- Do it until matrix A becomes a reduced row echelon matrix.
- The identity matrix on the right side has now become the inverse of matrix A and matrix A becomes an identity matrix.
- If while doing these steps we found a zero row, this means matrix A doesn't have an inverse.

For example:

$$A = \begin{bmatrix} 1 & 0 & 4 \\ 1 & 1 & 6 \\ -3 & 0 & -10 \end{bmatrix}$$

We will find the inverse of matrix A using elementary row operations and identity matrix.

$$\begin{pmatrix} 1 & 0 & 4 & | & 1 & 0 & 0 \\ 1 & 1 & 6 & | & 0 & 1 & 0 \\ -3 & 0 & -10 & | & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 4 & | & 1 & 0 & 0 \\ 0 & 1 & 2 & | & -1 & 1 & 0 \\ 0 & 0 & 2 & | & 3 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 4 & | & 1 & 0 & 0 \\ 0 & 1 & 2 & | & -1 & 1 & 0 \\ 0 & 0 & 1 & | & \frac{3}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & | & -5 & 0 & -2 \\ 0 & 1 & 0 & | & -4 & 1 & -1 \\ 0 & 0 & 1 & | & \frac{3}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

Therefore, we found the inverse of matrix A is :

$$A^{-1} = \begin{bmatrix} -5 & 0 & -2 \\ -4 & 1 & -1 \\ \frac{3}{2} & 0 & \frac{1}{2} \end{bmatrix}$$

D. System of Linear Equation

A system of linear equations occurs when we have multiple equations that need to be solved together. Each equation can be interpreted as a straight line in 2D or a plane in 3D, and solving the system means finding where these lines or planes intersect.

A system of linear equations with n variables can be denoted as:

$$\begin{gathered} a_{11}x_1 + a_{12}x_2 + \ldots\ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots\ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_nx_2 + \ldots\ldots + a_{nn}x_n = b_n \end{gathered}$$

Using matrix multiplication, we can write the system above as a matrix equation of

$$Ax = B$$

Where:
- A is a matrix with dimension of n×n
- x is a matrix with dimension of n×1
- B is a matrix with dimension of n×1

For example:
Using this equation

$$\begin{gathered} x_1 + 5x_2 = 2 \\ 12x_1 + 2x_2 = 11 \end{gathered}$$

We can write it in the form of matrix equations

$$\begin{bmatrix} 1 & 5 \\ 12 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 11 \end{bmatrix}$$

We will be solving this using Gauss-Jordan, first, we will change it to the form that is operable for using Gauss-Jordan.

$$\begin{bmatrix} 1 & 5 & | & 2 \\ 12 & 2 & | & 11 \end{bmatrix}$$

Then using Gauss-Jordan

$$\begin{bmatrix} 1 & 5 & | & 2 \\ 0 & -58 & | & -13 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 5 & | & 2 \\ 0 & 1 & | & \frac{13}{58} \end{bmatrix}$$

$$\left[\begin{array}{cc|c} 1 & 0 & \frac{51}{58} \\ 0 & 1 & \frac{13}{58} \end{array}\right]$$

Then we found the solution of this particular system of linear equations are

$$x_1 = \frac{51}{58}, \ x_2 = \frac{13}{58}$$

III. Problem Analysis

A. Mathematical Foundation

1. Expected Payoffs
   - Player 1's expected payoff: x^TAy.
   - Player 2's expected payoff: x^TBy
2. Nash Equilibrium Conditions

   A mixed strategy pair (x^*,y^*) is a Nash equilibrium if
   - $x^TAy^* \leq x^{*T}Ay^*$ for all x.
   - $x^{*T}By \leq x^{*T}By^*$ for all y.

B. Gauss-Jordan Approach

1. Transform to Systems of Equations
   - Player 1's strategy condition: $Ay^* \leq 1v$, where v is the value of the game for player 1.
   - Player 2's strategy condition: $x^{*T}B \leq 1w$, where w is the value of the game for player 2.
   - Mixed strategy constraint:
     1. $x^T1 = 1$(Player 1's strategies sum to 1)
     2. $y^T1 = 1$(Player 2's strategies sum to 1)
     3. $x^*, y^* \geq 0$(Non-negativity of probabilities)

   The equilibrium conditions can now be rewritten in a matrix form, with the goal of solving for the mixed strategies x^* and $y^*$and the game values v and w.
2. Matrix System
   1. Player 1's
      - $Ay^* = 1v$
      - $x^T1 = 1$
      - $x^* \geq 0$
   2. Player 2's
      - $x^{*T}B = 1w$
      - $y^T1 = 1$
      - $y^* \geq 0$
3. Solving using Gauss-Jordan

   The Gauss-Jordan method is an algorithm used to find the inverse of a matrix or solve a system of linear equations. This method works through the following steps:
   1. Constructing Augmented Matrix

      For both players 1 and 2, we construct the augmented matrix combining the payoff matrices A and B, along with the constraints and game values v and w
   2. Row Operations

      Use ERO to reduce the augmented matrix to row echelon form and eventually to reduced row echelon form (RREF).
   3. Extracting Solutions

      After the system is in RREF, we can extract the values for x^*, y^*,v, and w, which will represent the Nash equilibrium strategies for both players.

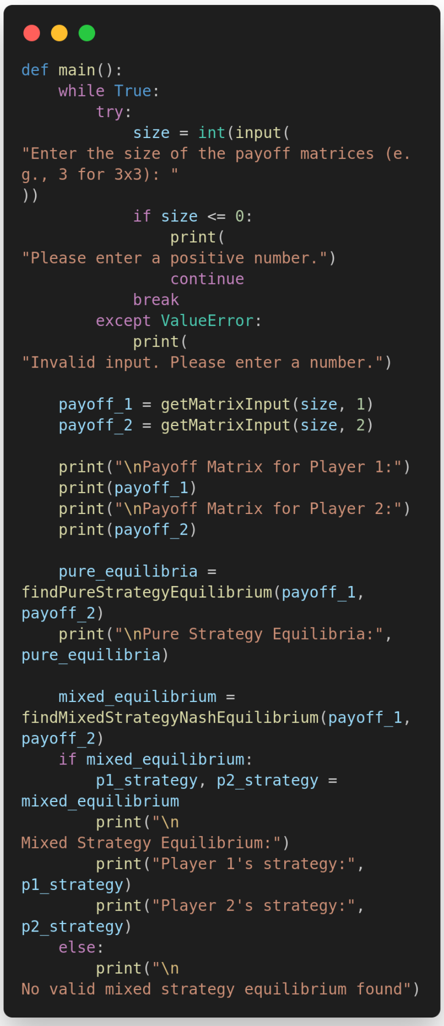
IV. Implementation

We will implement the program using Python. The program entry point is the main function that is executed upon starting the program.

The version of Python used is 3.12.3. We will only be using one library, *numpy*.

All functions regarding the functionality, its implementation, and purpose will be listed in the following section :

1. main

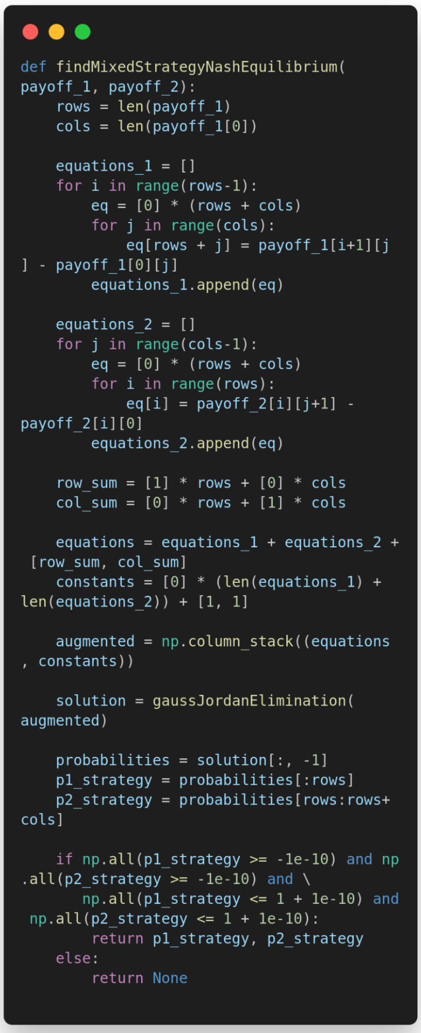


```
def main():
    while True:
        try:
            size = int(input(
"Enter the size of the payoff matrices (e.
g., 3 for 3x3): "
))
            if size <= 0:
                print(
"Please enter a positive number.")
                continue
            break
        except ValueError:
            print(
"Invalid input. Please enter a number.")

    payoff_1 = getMatrixInput(size, 1)
    payoff_2 = getMatrixInput(size, 2)

    print("\nPayoff Matrix for Player 1:")
    print(payoff_1)
    print("\nPayoff Matrix for Player 2:")
    print(payoff_2)

    pure_equilibria =
findPureStrategyEquilibrium(payoff_1,
payoff_2)
    print("\nPure Strategy Equilibria:",
pure_equilibria)

    mixed_equilibrium =
findMixedStrategyNashEquilibrium(payoff_1,
payoff_2)
    if mixed_equilibrium:
        p1_strategy, p2_strategy =
mixed_equilibrium
        print("\n
Mixed Strategy Equilibrium:")
        print("Player 1's strategy:",
p1_strategy)
        print("Player 2's strategy:",
p2_strategy)
    else:
        print("\n
No valid mixed strategy equilibrium found")
```

figure 4.1 main function

This is the main function that will accept our inputs as two matrices of the payoff matrix and then will call all the necessary functions to find the Pure and Mixed Nash Equilibrium.

2. findMixedStrategyNashEquilibrium

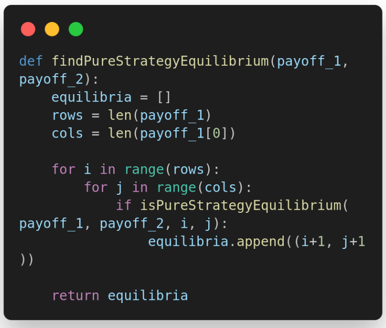


```python
def findMixedStrategyNashEquilibrium(
payoff_1, payoff_2):
    rows = len(payoff_1)
    cols = len(payoff_1[0])

    equations_1 = []
    for i in range(rows-1):
        eq = [0] * (rows + cols)
        for j in range(cols):
            eq[rows + j] = payoff_1[i+1][j
] - payoff_1[0][j]
        equations_1.append(eq)

    equations_2 = []
    for j in range(cols-1):
        eq = [0] * (rows + cols)
        for i in range(rows):
            eq[i] = payoff_2[i][j+1] -
payoff_2[i][0]
        equations_2.append(eq)

    row_sum = [1] * rows + [0] * cols
    col_sum = [0] * rows + [1] * cols

    equations = equations_1 + equations_2 +
 [row_sum, col_sum]
    constants = [0] * (len(equations_1) +
len(equations_2)) + [1, 1]

    augmented = np.column_stack((equations
, constants))

    solution = gaussJordanElimination(
augmented)

    probabilities = solution[:, -1]
    p1_strategy = probabilities[:rows]
    p2_strategy = probabilities[rows:rows+
cols]

    if np.all(p1_strategy >= -1e-10) and np
.all(p2_strategy >= -1e-10) and \
       np.all(p1_strategy <= 1 + 1e-10) and
 np.all(p2_strategy <= 1 + 1e-10):
        return p1_strategy, p2_strategy
    else:
        return None
```

figure 4.2 findMixedStrategyNashEquilibrium

This is the function responsible for finding our Mixed Strategy solution, first, it will construct the augmented matrix, then call the gaussJordanElimination function to find the solution.

3. findPureStrategyNashEquilibrium

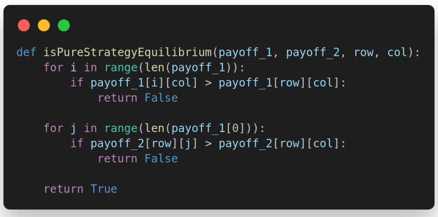


```python
def findPureStrategyEquilibrium(payoff_1,
payoff_2):
    equilibria = []
    rows = len(payoff_1)
    cols = len(payoff_1[0])

    for i in range(rows):
        for j in range(cols):
            if isPureStrategyEquilibrium(
payoff_1, payoff_2, i, j):
                equilibria.append((i+1, j+1
))

    return equilibria
```

figure 4.2 findPureStrategyNashEquilibrium

This is the function responsible for finding the Pure Strategy solution, we use isPureStrategyEquilibrium function to determine if a set of strategies is indeed a Pure Strategy solution.

4. isPureStrategyNashEquilibrium

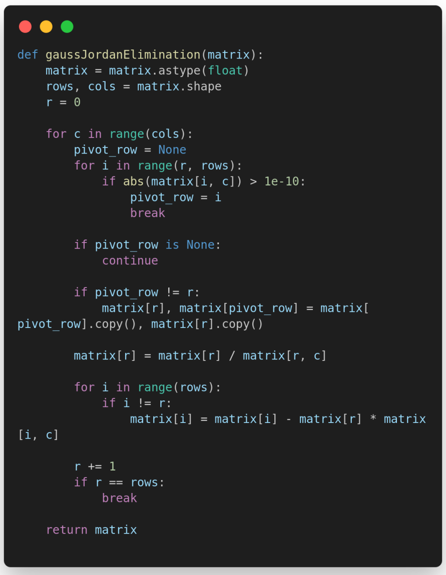


```python
def isPureStrategyEquilibrium(payoff_1, payoff_2, row, col):
    for i in range(len(payoff_1)):
        if payoff_1[i][col] > payoff_1[row][col]:
            return False

    for j in range(len(payoff_1[0])):
        if payoff_2[row][j] > payoff_2[row][col]:
            return False

    return True
```

figure 4.4 isPureStrategyNashEquilibrium

This is the function responsible for determining if a set of strategies is indeed a Pure Strategy solution.

5. gaussJordanElimination

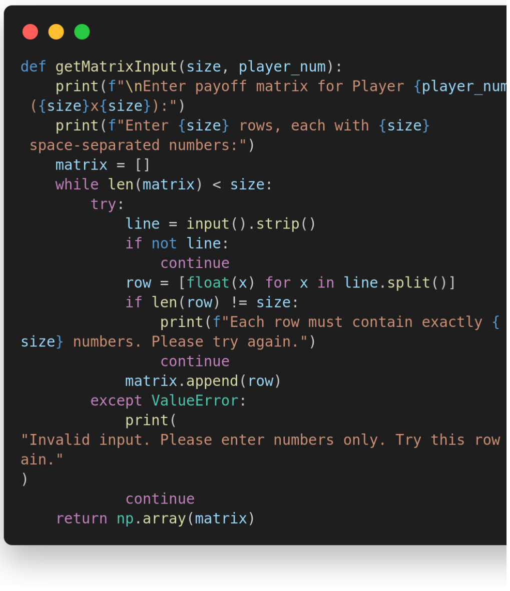


```python
def gaussJordanElimination(matrix):
    matrix = matrix.astype(float)
    rows, cols = matrix.shape
    r = 0

    for c in range(cols):
        pivot_row = None
        for i in range(r, rows):
            if abs(matrix[i, c]) > 1e-10:
                pivot_row = i
                break

        if pivot_row is None:
            continue

        if pivot_row != r:
            matrix[r], matrix[pivot_row] = matrix[
pivot_row].copy(), matrix[r].copy()

        matrix[r] = matrix[r] / matrix[r, c]

        for i in range(rows):
            if i != r:
                matrix[i] = matrix[i] - matrix[r] * matrix
[i, c]

        r += 1
        if r == rows:
            break

    return matrix
```

figure 4.5 gaussJordanElimination

This is the function responsible for finding the solution from an augmented matrix from the result of player 1 and player 2 game value and the constraint, it solves the augmented matrix using the Gauss-Jordan method

6. getMatrixInput



```
def getMatrixInput(size, player_num):
    print(f"\nEnter payoff matrix for Player {player_num
 ({size}x{size}):")
    print(f"Enter {size} rows, each with {size}
 space-separated numbers:")
    matrix = []
    while len(matrix) < size:
        try:
            line = input().strip()
            if not line:
                continue
            row = [float(x) for x in line.split()]
            if len(row) != size:
                print(f"Each row must contain exactly {
size} numbers. Please try again.")
                continue
            matrix.append(row)
        except ValueError:
            print(
"Invalid input. Please enter numbers only. Try this row
ain."
)
            continue
    return np.array(matrix)
```

figure 4.6 getMatrixInput

This is the function responsible for accepting the input of two matrices which are the payoff matrix for each player 1 and player 2.

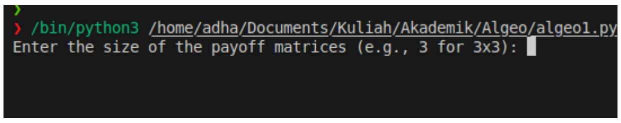
V. Testing and Analysis

The program could be executed by simply running the Python file, making sure the required Python and its libraries are properly installed.

For the testing, we will be using a 3x3 payoff matrix for each player, Player 1's payoff matrix is A, and Player 2's payoff matrix is B

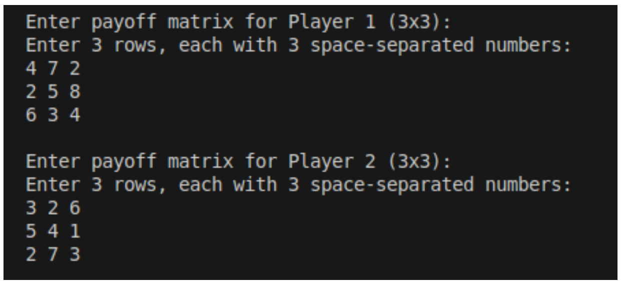
$$A = \begin{bmatrix} 4 & 7 & 2 \\ 2 & 5 & 8 \\ 6 & 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 2 & 6 \\ 5 & 4 & 1 \\ 2 & 7 & 3 \end{bmatrix}$$

The following set of figures demonstrates the execution of the Nash equilibrium solver program as a whole.



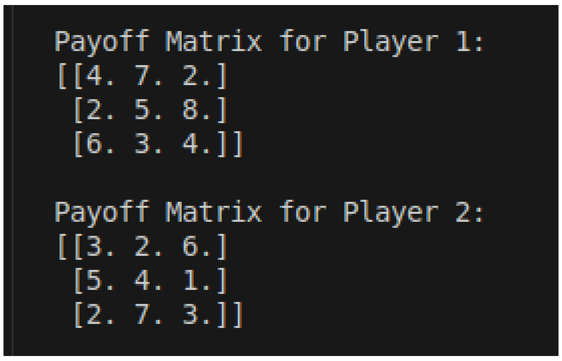
```
❯
❯ /bin/python3 /home/adha/Documents/Kuliah/Akademik/Algeo/algeo1.py
Enter the size of the payoff matrices (e.g., 3 for 3x3): 
```

figures 5.1 Size of Matrix Input



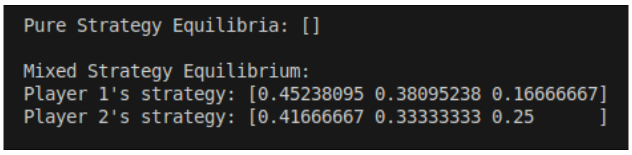
```
Enter payoff matrix for Player 1 (3x3):
Enter 3 rows, each with 3 space-separated numbers:
4 7 2
2 5 8
6 3 4

Enter payoff matrix for Player 2 (3x3):
Enter 3 rows, each with 3 space-separated numbers:
3 2 6
5 4 1
2 7 3
```

figures 5.2 Payoff Matrices Input



```
Payoff Matrix for Player 1:
[[4. 7. 2.]
 [2. 5. 8.]
 [6. 3. 4.]]

Payoff Matrix for Player 2:
[[3. 2. 6.]
 [5. 4. 1.]
 [2. 7. 3.]]
```

figures 5.3 Inputted Payoff Marix



```
Pure Strategy Equilibria: []

Mixed Strategy Equilibrium:
Player 1's strategy: [0.45238095 0.38095238 0.16666667]
Player 2's strategy: [0.41666667 0.33333333 0.25      ]
```

figures 5.4 Nash Equilibrium Result

Through the testing, it can be concluded that the process of finding Nash equilibrium, either pure or mixed has proven successful.

However, this testing also uncovers some notable drawbacks that should be considered:

Firstly, by using Gauss-Jordan to find the Nash equilibrium, we are prone to rounding errors because of the operation. This could potentially impact the accuracy, especially for large matrix dimensions.

Secondly, by using Gauss-Jordan to find the Nash equilibrium, for cases where big matrix dimensions are used, the program shows drawbacks in terms of speed when doing ERO. This could potentially impact performance when computing a matrix with large dimensions.

Thirdly, using Gauss-Jordan poses a risk where there is more than one solution, making the program miss other Nash equilibrium solutions.

Lastly, there is a recognized need for a more reliable way to compute large dimensions matrix. The current method may pose a risk of accuracy and speed when computing large dimension matrix.

VI. Conclusion

In conclusion, implementing Gauss-Jordan for finding the Nash Equilibrium for the Bimatrix Games has demonstrated both significant capabilities and notable limitations. The method successfully identifies mixed strategy Nash equilibrium, proving a fundamental tool for game-theoretical analysis. However, several key challenges emerge when dealing with larger matrices: the accumulation of rounding errors during Gauss-Jordan operations can compromise accuracy, and the computational overhead of elementary row operations leads to decreased performance with larger dimensions. These limitations underscore the need for more robust computational methods when handling large-scale game matrices.

VII. Acknowledgment

The completion of this paper could not have been possible without the aid of all IF2123 lecturers, especially Dr. Ir. Rinaldi Munir who has taught the K02 for the linear and geometry algebra. The writer has learned a hefty amount of information in the process of developing this paper.

References

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-02-Matriks-Eselon-2023.pdf, diakses pada 26 Desember 2024.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf, dikases pada 26 Desember 2024.

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-05-Sistem-Persamaan-Linier-2-2023.pdf, diakses pada 26 Desember 2024.

[4] C. E. Lemke, J. T. Howson, "Equilibrium Points of Bimatrix Games," Journal of the Society for Industrial and Applied Mathematics, Vol. 12, No. 2, 1964, pp. 413-423.

[5] M.J. Osborne, “An Introduction to Game Theory,” Oxford University Press, 2000.

Statement

Hereby, I declare this paper I have written with my own work, not a reproduction or translation of someone else’s paper, and not plagiarized.

Bandung, 28 Desember 2024

[illegible]

Muhammad Adha Ridwan 13523098