Прізвище: Колісник
Ім'я: Андрій
Група: КН-405
Кафедра.: Кафедра Систем Автоматизованого Проектування
Дисципліна: Теорія прийняття рішень
Перевірив: Кривий Р.З.

Звіт
До лабораторної роботи №5
На тему "Матричні ігри"

Мета роботи: Визначити основні поняття теорії ігор, властивості змішаних стратегій. Вивчити метод вирішення матричних ігор у змішаних стратегіях за допомогою введення до подвійних завдань лінійного програмування.

Короткі теоретичні відомості

У грі беруть участь два гравці: А і В. У розпорядженні кожного гравця є кінцеве безліч варіантів вибору - стратегій. Нехай - безліч стратегій гравця А, - безліч стратегій гравця В. З кожною парою стратегій пов'язаний платіж, який один з гравців виплачує іншому. Тобто, коли гравець А вибирає стратегію (свою і-ю стратегію), а гравець В - стратегію, то результатом такого вибору стає платіж. Оскільки стратегій кінцеве число, то платежі утворюють матрицю розмірності n x m, звану матрицею платежів (або матрицею гри). Рядки цієї матриці відповідають стратегіям гравця А, а стовпці - стратегіям гравця В.

Порядок вирішення завдання:

1) Вихідні дані беруть із варіантів індивідуальних завдань.
2) При вирішенні матричної гри потрібно вийти на наступні етапи:
    1. Знайти сідлову точку і перевірити, чи має гра вирішення в чистих стратегій.
    2. У випадку відсутності чистої стратегії, знайти рішення в оптимальних змішаних стратегіях
    3. Спростити платіжну матрицю (перевірити матрицю на домінуючі рядки і стовбці).
    4. Визначити оптимальні плани за допомогою одного з методів лінійного програмування.
    5. Знайдіть рішення гри.

Індивідуальне завдання (варіант 13):

11;7;10;8;12
14;7;11;9;15
11;7;11;15;10
14;11;9;12;11
15;10;7;8;10

Виконання:

Дані для матричної гри зчитуються з зовнішнього файлу (Рис. 1)

```
kolisnyk9@MacBook-Pro-Andrew Lab5 % cat text.txt
11;7;10;8;12
14;7;11;9;15
11;7;11;15;10
14;11;9;12;11
15;10;7;8;10
kolisnyk9@MacBook-Pro-Andrew Lab5 %
```

Рис. 1 Файл text.txt

Після отримання даних першим кроком є перевірка існування сідлової точки в платіжній матриці (рис. 1).

```
Початкові дані
+----+----+----+----+----+----+
|    | B1 | B2 | B3 | B4 | B5 |
+----+----+----+----+----+----+
| A1 | 11 | 7  | 10 | 8  | 12 |
| A2 | 14 | 7  | 11 | 9  | 15 |
| A3 | 11 | 7  | 11 | 15 | 10 |
| A4 | 14 | 11 | 9  | 12 | 11 |
| A5 | 15 | 10 | 7  | 8  | 10 |
+----+----+----+----+----+----+
Перевірка матриці на сідлову точку
+-------------+----+----+----+----+----+--------------+
|             | B1 | B2 | B3 | B4 | B5 | a = min(Ai) |
+-------------+----+----+----+----+----+--------------+
|      A1     | 11 | 7  | 10 | 8  | 12 |      7       |
|      A2     | 14 | 7  | 11 | 9  | 15 |      7       |
|      A3     | 11 | 7  | 11 | 15 | 10 |      7       |
|      A4     | 14 | 11 | 9  | 12 | 11 |      9       |
|      A5     | 15 | 10 | 7  | 8  | 10 |      7       |
| b = max(Bi) | 15 | 11 | 11 | 15 | 15 |              |
+-------------+----+----+----+----+----+--------------+
a = max(min(Ai)) = 9
b = min(max(Bi)) = 11
Сідлова точка відсутня, так як a != b
Ціна гри знаходиться в межах:  9 <= y <= 11
```

Рис. 1 Перевірка сідлової точки

Так як сідлова точка відсутня, то перевіряємо матрицю на домінуючі стовпці і домінуючі рядки (рис. 2)

```
Перевіряємо матрицю на домінуючі рядки і домінуючі стовпці:
З позиції виграшу гравця A
Стретегія A2 домінуюча над стратегією A1 тому забираємо рядок 1
Після перевірки домінуючих рядків наша матриця набула наступного вигляду:
+----+----+----+----+----+----+
|    | B1 | B2 | B3 | B4 | B5 |
+----+----+----+----+----+----+
| A1 | 14 | 7  | 11 | 9  | 15 |
| A2 | 11 | 7  | 11 | 15 | 10 |
| A3 | 14 | 11 | 9  | 12 | 11 |
| A4 | 15 | 10 | 7  | 8  | 10 |
+----+----+----+----+----+----+
З позиції програшу гравця B
Стретегія B2 домінуюча над стратегією B1 тому забираємо стовпчик 1
Стретегія B2 домінуюча над стратегією B5 тому забираємо стовпчик 5
Після перевірки домінуючих стовпчиків наша матриця набула наступного вигляду:
+----+----+----+----+
|    | B1 | B2 | B3 |
+----+----+----+----+
| A1 | 7  | 11 | 9  |
| A2 | 7  | 11 | 15 |
| A3 | 11 | 9  | 12 |
| A4 | 10 | 7  | 8  |
+----+----+----+----+
```

Рис. 2 Перевірка матриці на домінуючі стовпці і домінуючі рядки

Далі отриману матрицю записуємо у вигляді задачі лінійного програмування (рис. 3)

```
Знаходимо рішення гри в змішаних стратегіях
Знайти мінімум функції F(x) при обмеженнях (для гравця ||)
7x_1 + 7x_2 + 11x_3 + 10x_4 >= 1
11x_1 + 11x_2 + 9x_3 + 7x_4 >= 1
9x_1 + 15x_2 + 12x_3 + 8x_4 >= 1
F(x) = x_1 + x_2 + x_3 + x_4 --> min
Знайти мінімум функції Z(y) при обмеженнях (для гравця |)
7y_1 + 11y_2 + 9y_3 <= 1
7y_1 + 11y_2 + 15y_3 <= 1
11y_1 + 9y_2 + 12y_3 <= 1
10y_1 + 7y_2 + 8y_3 <= 1
Z(y) = 1y_1 + 1y_2 + 1y_3 --> max

Вирішимо пряму задачу лінійного програмування симплексним методом
Визначимо максимальне значення цільової функції  1y_1 + 1y_2 + 1y_3 --> max при наступних умовах-обмеженнях:
```

Рис. 3 Запис матриці у вигляді задачі лінійного програмування

Для розв’язання задачі лінійного програмування буде використовуватися симплекс метод. На рис. 4 зображено початкову симплекс таблицю.

Початкова симплекс-таблиця:

Базис	y1	y2	y3	y4	y5	y6	y7	B
y4	7	11	9	1	0	0	0	1
y5	7	11	15	0	1	0	0	1
y6	11	9	12	0	0	1	0	1
y7	10	7	8	0	0	0	1	1
Z(y)	-1	-1	-1	0	0	0	0	0

Рис. 4 Початкова симплекс таблиця

На рис. 5 зображено основний алгоритм симплекс методу.

Далі покроково показано розв'язання симплекс методом:

Базис	y1	y2	y3	y4	y5	y6	y7	B
y4	0	58/11	15/11	1	0	-7/11	0	4/11
y5	0	58/11	81/11	0	1	-7/11	0	4/11
y6	1	9/11	12/11	0	0	1/11	0	1/11
y7	0	-13/11	-32/11	0	0	-10/11	1	1/11
Z(y)	0	-2/11	1/11	0	0	1/11	0	1/11

Базис	y1	y2	y3	y4	y5	y6	y7	B
y4	0	1	15/58	11/58	0	-7/58	0	2/29
y5	0	0	6	-1	1	0	0	0
y6	1	0	51/58	-9/58	0	11/58	0	1/29
y7	0	0	-151/58	13/58	0	-61/58	1	5/29
Z(y)	0	0	4/29	1/29	0	2/29	0	3/29

Кінцева симплекс-таблиця має наступний вигляд:

Базис	y1	y2	y3	y4	y5	y6	y7	B
y4	0	1	15/58	11/58	0	-7/58	0	2/29
y5	0	0	6	-1	1	0	0	0
y6	1	0	51/58	-9/58	0	11/58	0	1/29
y7	0	0	-151/58	13/58	0	-61/58	1	5/29
Z(y)	0	0	4/29	1/29	0	2/29	0	3/29

Рис. 5 Основний алгоритм симплекс методу

Результат, який вийшов після розв’язання задачі лінійного програмування можна побачити на рис. 6

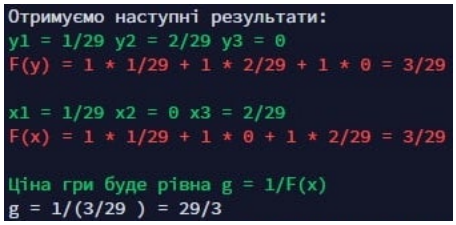

Рис. 6 Результати розв'язання

З отриманих результатів можемо зробити висновок, що ціна гри рівна 29/3

Повний код програми знаходиться за посиланням:
https://github.com/Kolisnyk9/Lab

Код програми:

Файл main.py

```
import sys
import re
import string
from copy import deepcopy
from colorama import Fore, Style, Back
from prettytable import PrettyTable
from simplex import Simplex

def open_file():
    if (len(sys.argv) < 2):
        print ('Enter file name')
        exit()

    fileName = sys.argv[1]
    try:
        return open(fileName)
    except FileNotFoundError:
        print("Oops! File not exist...")
        exit()


def get_matrix_table(matrix):
    if len(matrix) < 1: return ''

    x = PrettyTable()
    fields = ['']
    for i in range(len(matrix[0])):
        fields.append("B" + str(i + 1))

    x.field_names = fields
    for i in range(len(matrix)):
        x.add_row(['A' + str(i + 1)] + matrix[i])
    return x


def check_saddle_point(minA, maxB):
    maxFromMatrixA = minA[max(minA, key = minA.get)]
    minFromMatrixB = maxB[min(maxB, key = maxB.get)]
    return [maxFromMatrixA, minFromMatrixB]


def check_rows(firstRow, secondRow):
    equalElements = 0
    for i in range(len(firstRow)):
        if firstRow[i] < secondRow[i]: return 0
        if firstRow[i] == secondRow[i]: equalElements += equalElements

    return 0 if equalElements == len(firstRow) else 1
```

```
def check_dominant_rows(matrix):
    matrixAfterExcludingRows = []
    deletedRows = []

    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if i == j: continue
            result = check_rows(matrix[i], matrix[j])
            if result != 0 and j not in deletedRows:
                deletedRows.append(j)
                print("Стретегія А" + str(i + 1), "домінуюча над стратегією А" + str(j +
1), "тому забираємо рядок", j + 1)

    for i in range(len(matrix)):
        if i in deletedRows: continue
        matrixAfterExcludingRows.append(matrix[i])

    return matrixAfterExcludingRows


def check_dominant_columns(matrix):
    matrixAfterExcludingColumns = []
    deletedColumns = []
    transposedMatrix = [list(x) for x in zip(*matrix)]

    for i in range(len(transposedMatrix)):
        for j in range(len(transposedMatrix)):
            if i == j: continue
            result = check_rows(transposedMatrix[j], transposedMatrix[i])
            if result != 0 and j not in deletedColumns:
                deletedColumns.append(j)
                print("Стретегія В" + str(i + 1), "домінуюча над стратегією В" + str(j +
1), "тому забираємо стовпчик", j + 1)

    for i in range(len(matrix)):
        matrixAfterExcludingColumns.append([])
        for j in range(len(matrix[i])):
            if j in deletedColumns: continue
            matrixAfterExcludingColumns[i].append(matrix[i][j])

    return matrixAfterExcludingColumns


file = open_file()

matrix = []
for line in file:
    matrix.append([int(d) for d in re.split(';', re.sub('\n', '', line))])

if len(matrix) < 2: exit()
print(Fore.WHITE + "Початкові дані", Style.RESET_ALL)
print(get_matrix_table(matrix))

print(Fore.WHITE + "Перевірка матриці на сідлову точку", Style.RESET_ALL)
x = PrettyTable()
fields = ['']
for i in range(len(matrix)):
    fields.append("B" + str(i + 1))

fields.append('a = min(Ai)')
x.field_names = fields

minA = {}
```

```
maxB = {}

for i in range(len(matrix)):
    minA[i] = min(matrix[i])
    x.add_row(['A' + str(i + 1)] + matrix[i] + [Fore.WHITE + str(minA[i]) +
Style.RESET_ALL])
    for j in range(len(matrix[i])):
        if j not in maxB or maxB[j] < matrix[i][j]:
            maxB[j] = matrix[i][j]

x.add_row(['b = max(Bi)' + Fore.WHITE] + [maxB[element] for element in maxB] + ['' +
Style.RESET_ALL])
print(x)

[maxFromMatrixA, minFromMatrixB] = check_saddle_point(minA, maxB)
if maxFromMatrixA == minFromMatrixB:
    print("Сідлова точка присутня!")
else:
    print("a = max(min(Ai)) =", maxFromMatrixA)
    print("b = min(max(Bi)) =", minFromMatrixB)
    print("Сідлова точка відсутня, так як a != b")
    print("Ціна гри знаходиться в межах:", Fore.RED, maxFromMatrixA, "<= y <=",
minFromMatrixB, Style.RESET_ALL)


print(Fore.WHITE, "\nПеревіряємо матрицю на домінуючі рядки і домінуючі стовпці:",
Style.RESET_ALL)
print(Fore.WHITE + "З позиції виграшу гравця A", Style.RESET_ALL)
matrixAfterExcludingRows = check_dominant_rows(matrix)
print(Fore.RED + "Після перевірки домінуючих рядків наша матриця набула наступного
вигляду: ", Style.RESET_ALL)
print(get_matrix_table(matrixAfterExcludingRows))


print(Fore.WHITE + "З позиції програшу гравця B", Style.RESET_ALL)
matrixAfterExcludingColumns = check_dominant_columns(matrixAfterExcludingRows)
print(Fore.RED + "Після перевірки домінуючих стовпчиків наша матриця набула наступного
вигляду: ", Style.RESET_ALL)
print(get_matrix_table(matrixAfterExcludingColumns))

transposedMatrix = [list(x) for x in zip(*matrixAfterExcludingColumns)]
print(Fore.WHITE + "\nЗнаходимо рішення гри в змішаних стратегіях", Style.RESET_ALL)
print(Fore.RED + "Знайти мінімум функції F(x) при обмеженнях (для гравця ||)",
Style.RESET_ALL)

secondPlayersConditions = []
for i in range(len(transposedMatrix)):
    secondPlayersConditions.append('')
    for j in range(len(transposedMatrix[i])):
        secondPlayersConditions[i] += str(transposedMatrix[i][j]) + 'x_' + str(j + 1) +
' + '

for i in range(len(secondPlayersConditions)):
    print(secondPlayersConditions[i][:-2] + '>= 1')

mainCondition = 'F(x) = '
for i in range(len(matrixAfterExcludingColumns)):
    mainCondition += 'x_' + str(i + 1) + ' + '

print(mainCondition[:-2] + '--> min')

print(Fore.RED + "Знайти мінімум функції Z(y) при обмеженнях (для гравця |)",
Style.RESET_ALL)
firstPlayersConditions = []
```

```
vars_count = 0
for i in range(len(matrixAfterExcludingColumns)):
    firstPlayersConditions.append('')
    columns = len(matrixAfterExcludingColumns[i])
    for j in range(columns):
        firstPlayersConditions[i] += str(matrixAfterExcludingColumns[i][j]) + 'y_' +
str(j + 1) + ' + '
        if columns > vars_count:
            vars_count = columns

conditions = ''
for i in range(len(firstPlayersConditions)):
    firstPlayersConditions[i] = firstPlayersConditions[i][:-2] + '<= 1'

mainCondition = ''
for i in range(len(transposedMatrix)):
    mainCondition += '1y_' + str(i + 1) + ' + '

mainCondition = mainCondition[:-2]

for line in firstPlayersConditions:print(line)
print('Z(y) = ' + mainCondition + '--> max')

print(Fore.WHITE + "\nВирішимо пряму задачу лінійного програмування симплексним методом",
Style.RESET_ALL)
print(Fore.WHITE + "Визначимо максимальне значення цільової функції", Fore.RED,
mainCondition + '--> max', Fore.WHITE + "при наступних умовах-обмеженнях:",
Style.RESET_ALL)
print(conditions)
print(Fore.WHITE + "Після переведення в канонічну форму переходимо до основно алгоритму
симплекс-методом", Style.RESET_ALL)

simplexResult = Simplex(num_vars=vars_count, constraints=firstPlayersConditions,
objective_function=mainCondition)
print(Fore.WHITE + "\nОтримуємо наступні результати:", Style.RESET_ALL)

x_result = {}
y_result = {}
for key in simplexResult.solution:
    if 'y_' in key:
        y_result[key] = simplexResult.solution[key]
    elif 'x_' in key:
        x_result[key] = simplexResult.solution[key]

yResultCond = 'F(y) = '
yResult = 0
for i in range(vars_count):
    print('y' + str(i + 1) + ' =', y_result['y_' + str(i + 1)], end=' ')
    yResult += 1 * y_result['y_' + str(i + 1)]
    yResultCond += "1 * " + str(y_result['y_' + str(i + 1)]) + ' + '


print("\n" + Fore.RED + yResultCond[:-2] + '= ' + str(yResult), Style.RESET_ALL, '\n')

xResultCond = 'F(x) = '
xResult = 0
for i in range(vars_count):
    print('x' + str(i + 1) + ' =', x_result['x_' + str(i + 1)], end=' ')
    xResult += 1 * x_result['x_' + str(i + 1)]
    xResultCond += "1 * " + str(x_result['x_' + str(i + 1)]) + ' + '


print("\n" + Fore.RED + xResultCond[:-2] + '= ' + str(xResult), Style.RESET_ALL)
```

```
print("\nЦіна гри буде рівна g = 1/F(x)")
print(Fore.WHITE + "g = 1/(" + str(xResult), ") =", str(1/xResult), Style.RESET_ALL)
```

simple.py

```
from fractions import Fraction
from warnings import warn
from prettytable import PrettyTable
from colorama import Fore, Style, Back

class Simplex(object):
    def __init__(self, num_vars, constraints, objective_function):
        self.num_vars = num_vars
        self.constraints = constraints
        self.objective_function = objective_function
        self.coeff_matrix,    self.r_rows,    self.num_s_vars,    self.num_r_vars    =
self.construct_matrix_from_constraints()
        del self.constraints
        self.basic_vars = [0 for i in range(len(self.coeff_matrix))]
        self.phase1()
        r_index = self.num_r_vars + self.num_s_vars

        for i in self.basic_vars:
            if i > r_index:
                raise ValueError("Нездійсненне рішення")

        self.delete_r_vars()

        self.solution = self.objective_maximize()
        self.optimize_val = self.coeff_matrix[0][-1]

    def print_simplex_table(self):
        x = PrettyTable()
        x.field_names   =   ['Базис']   +   ['y'   +   str(i   +   1)   for   i   in
range(len(self.coeff_matrix[0]) - 1)] + ['B']
        for i in range(1, len(self.coeff_matrix)):
            x.add_row(['y' + str(i + self.num_vars)] + self.coeff_matrix[i])

        x.add_row(['Z(y)'] + self.coeff_matrix[0])
        print(x)


    def construct_matrix_from_constraints(self):
        num_s_vars = 0
        num_r_vars = 0
        for expression in self.constraints:
            if '>=' in expression:
                num_s_vars += 1

            elif '<=' in expression:
                num_s_vars += 1
                num_r_vars += 1

            elif '=' in expression:
                num_r_vars += 1
        total_vars = self.num_vars + num_s_vars + num_r_vars

        coeff_matrix  =  [[Fraction("0/1")  for  i  in  range(total_vars+1)]  for  j  in
range(len(self.constraints)+1)]
        s_index = self.num_vars
        r_index = self.num_vars + num_s_vars
        r_rows = []
        for i in range(1, len(self.constraints)+1):
            constraint = self.constraints[i-1].split(' ')

            for j in range(len(constraint)):

                if '_' in constraint[j]:
```

```
                coeff, index = constraint[j].split('_')
                if constraint[j-1] == '-':
                    coeff_matrix[i][int(index)-1] = Fraction("-" + coeff[:-1] + "/1")
                else:
                    coeff_matrix[i][int(index)-1] = Fraction(coeff[:-1] + "/1")

            elif constraint[j] == '<=':
                coeff_matrix[i][s_index] = Fraction("1/1")
                s_index += 1

            elif constraint[j] == '>=':
                coeff_matrix[i][s_index] = Fraction("-1/1")
                coeff_matrix[i][r_index] = Fraction("1/1")
                s_index += 1
                r_index += 1
                r_rows.append(i)

            elif constraint[j] == '=':
                coeff_matrix[i][r_index] = Fraction("1/1")
                r_index += 1
                r_rows.append(i)

        coeff_matrix[i][-1] = Fraction(constraint[-1] + "/1")


    return coeff_matrix, r_rows, num_s_vars, num_r_vars

def phase1(self):
    r_index = self.num_vars + self.num_s_vars
    for i in range(r_index, len(self.coeff_matrix[0])-1):
        self.coeff_matrix[0][i] = Fraction("-1/1")
    coeff_0 = 0
    for i in self.r_rows:
        self.coeff_matrix[0] = add_row(self.coeff_matrix[0], self.coeff_matrix[i])
        self.basic_vars[i] = r_index
        r_index += 1
    s_index = self.num_vars
    for i in range(1, len(self.basic_vars)):
        if self.basic_vars[i] == 0:
            self.basic_vars[i] = s_index
            s_index += 1

    key_column = max_index(self.coeff_matrix[0])
    condition = self.coeff_matrix[0][key_column] > 0

    while condition is True:

        key_row = self.find_key_row(key_column = key_column)
        self.basic_vars[key_row] = key_column
        pivot = self.coeff_matrix[key_row][key_column]
        self.normalize_to_pivot(key_row, pivot)
        self.make_key_column_zero(key_column, key_row)

        key_column = max_index(self.coeff_matrix[0])
        condition = self.coeff_matrix[0][key_column] > 0

def find_key_row(self, key_column):
    min_val = float("inf")
    min_i = 0
    for i in range(1, len(self.coeff_matrix)):
        if self.coeff_matrix[i][key_column] > 0:
            val = self.coeff_matrix[i][-1] / self.coeff_matrix[i][key_column]
            if val <  min_val:
                min_val = val
```

```
                    min_i = i
        if min_val == float("inf"):
            raise ValueError("Необмежене рішення")
        if min_val == 0:
            warn("Error")
        return min_i

    def normalize_to_pivot(self, key_row, pivot):
        for i in range(len(self.coeff_matrix[0])):
            self.coeff_matrix[key_row][i] /= pivot

    def make_key_column_zero(self, key_column, key_row):
        num_columns = len(self.coeff_matrix[0])
        for i in range(len(self.coeff_matrix)):
            if i != key_row:
                factor = self.coeff_matrix[i][key_column]
                for j in range(num_columns):
                    self.coeff_matrix[i][j] -= self.coeff_matrix[key_row][j] * factor

    def delete_r_vars(self):
        for i in range(len(self.coeff_matrix)):
            non_r_length = self.num_vars + self.num_s_vars + 1
            length = len(self.coeff_matrix[i])
            while length != non_r_length:
                del self.coeff_matrix[i][non_r_length-1]
                length -= 1

    def update_objective_function(self):
        objective_function_coeffs = self.objective_function.split()
        for i in range(len(objective_function_coeffs)):
            if '_' in objective_function_coeffs[i]:
                coeff, index = objective_function_coeffs[i].split('_')
                if objective_function_coeffs[i-1] == '-':
                    self.coeff_matrix[0][int(index)-1] = Fraction(coeff[:-1] + "/1")
                else:
                    self.coeff_matrix[0][int(index)-1] = Fraction("-" + coeff[:-1] +
"/1")

    def objective_maximize(self):
        self.update_objective_function()

        for row, column in enumerate(self.basic_vars[1:]):
            if self.coeff_matrix[0][column] != 0:
                self.coeff_matrix[0]           =           add_row(self.coeff_matrix[0],
multiply_const_row(-self.coeff_matrix[0][column], self.coeff_matrix[row+1]))

        key_column = min_index(self.coeff_matrix[0])
        condition = self.coeff_matrix[0][key_column] < 0

        print(Fore.WHITE + "Початкова симплекс-таблиця:" + Style.RESET_ALL)
        self.print_simplex_table()

        print(Fore.WHITE + "\nДалі покроково показано розв'язання симплекс методом:" +
Style.RESET_ALL)
        while condition is True:
            key_row = self.find_key_row(key_column = key_column)
            self.basic_vars[key_row] = key_column
            pivot = self.coeff_matrix[key_row][key_column]
            self.normalize_to_pivot(key_row, pivot)
            self.make_key_column_zero(key_column, key_row)

            key_column = min_index(self.coeff_matrix[0])
            condition = self.coeff_matrix[0][key_column] < 0
            self.print_simplex_table()
```

```
        print("\n" + Fore.WHITE + 'Кінцева симплекс-таблиця має наступний вигляд:' +
Style.RESET_ALL)
        self.print_simplex_table()
        solution = {}
        for i, var in enumerate(self.basic_vars[1:]):
            if var < self.num_vars:
                solution['y_'+str(var+1)] = self.coeff_matrix[i+1][-1]

        for i in range(0, self.num_vars):
            if i not in self.basic_vars[1:]:
                solution['y_'+str(i+1)] = Fraction("0/1")

        count = 1
        for i in range(self.num_vars, len(self.coeff_matrix[0]) - 1):
            solution['x_' + str(count)] = self.coeff_matrix[0][i]
            count += 1

        return solution

def add_row(row1, row2):
    row_sum = [0 for i in range(len(row1))]
    for i in range(len(row1)):
        row_sum[i] = row1[i] + row2[i]
    return row_sum

def max_index(row):
    max_i = 0
    for i in range(0, len(row)-1):
        if row[i] > row[max_i]:
            max_i = i

    return max_i

def multiply_const_row(const, row):
    mul_row = []
    for i in row:
        mul_row.append(const*i)
    return mul_row

def min_index(row):
    min_i = 0
    for i in range(0, len(row)):
        if row[min_i] > row[i]:
            min_i = i

    return min_i
```

Висновок: в ході виконання лабораторної роботи було отримано теоретичні знання про матричці ігри, написано програму для вирішення матричної гри, яка шукає сідлову точку, спрощує матрицю шляхом знаходження домінуючих рядків та стовпці, та розв’язує отриману задачу лінійного програмування симплекс методом.