Лабораторная работа №2 по курсу Математическая экономика

Работу выполнил Гамов Павел м80-407б-18

In [1]:
```
from pulp import *
import numpy as np
```

In [2]:
```
k = 4
l = 7
n = 10 + k // 4
m = 30 - k // 4
```

1 пункт. Формируем матрицу A

In [3]:
```
f = lambda i,j: -k + ((1451*i + 1571*j + 2081*k + 2543 * l) % (30 + k // 5))
A = [[f(i,j) for j in range(n)] for i in range(m)]
A
```

Out[3]:
```
[[21, 2, 13, 24, 5, 16, -3, 8, 19, 0, 11],
 [2, 13, 24, 5, 16, -3, 8, 19, 0, 11, 22],
 [13, 24, 5, 16, -3, 8, 19, 0, 11, 22, 3],
 [24, 5, 16, -3, 8, 19, 0, 11, 22, 3, 14],
 [5, 16, -3, 8, 19, 0, 11, 22, 3, 14, 25],
 [16, -3, 8, 19, 0, 11, 22, 3, 14, 25, 6],
 [-3, 8, 19, 0, 11, 22, 3, 14, 25, 6, 17],
 [8, 19, 0, 11, 22, 3, 14, 25, 6, 17, -2],
 [19, 0, 11, 22, 3, 14, 25, 6, 17, -2, 9],
 [0, 11, 22, 3, 14, 25, 6, 17, -2, 9, 20],
 [11, 22, 3, 14, 25, 6, 17, -2, 9, 20, 1],
 [22, 3, 14, 25, 6, 17, -2, 9, 20, 1, 12],
 [3, 14, 25, 6, 17, -2, 9, 20, 1, 12, 23],
 [14, 25, 6, 17, -2, 9, 20, 1, 12, 23, 4],
 [25, 6, 17, -2, 9, 20, 1, 12, 23, 4, 15],
 [6, 17, -2, 9, 20, 1, 12, 23, 4, 15, -4],
 [17, -2, 9, 20, 1, 12, 23, 4, 15, -4, 7],
 [-2, 9, 20, 1, 12, 23, 4, 15, -4, 7, 18],
 [9, 20, 1, 12, 23, 4, 15, -4, 7, 18, -1],
 [20, 1, 12, 23, 4, 15, -4, 7, 18, -1, 10],
 [1, 12, 23, 4, 15, -4, 7, 18, -1, 10, 21],
 [12, 23, 4, 15, -4, 7, 18, -1, 10, 21, 2],
 [23, 4, 15, -4, 7, 18, -1, 10, 21, 2, 13],
 [4, 15, -4, 7, 18, -1, 10, 21, 2, 13, 24],
 [15, -4, 7, 18, -1, 10, 21, 2, 13, 24, 5],
 [-4, 7, 18, -1, 10, 21, 2, 13, 24, 5, 16],
 [7, 18, -1, 10, 21, 2, 13, 24, 5, 16, -3],
 [18, -1, 10, 21, 2, 13, 24, 5, 16, -3, 8],
 [-1, 10, 21, 2, 13, 24, 5, 16, -3, 8, 19]]
```

2 пункт. Найти гарантированные выигрыши первого и второго игрока при использовании чистых стратегий и их гарантирующие стратегии

Напишем свой минимакс и максимин для вычисления верхней и нижней цены игры.

In [4]:
```
def minmax(A, m, n):
    max_ = []
    for j in range(n):
        best = -10**100
        for i in range(m):
            if A[i][j] > best:
                best = A[i][j]
        max_.append(best)
    b = -1
    for k in range(n):
        if max_[k] == min(max_):
            b = k
    print("strategy: ", b)
    return min(max_)
```

Получается что максимальная цена игры равняется:

In [5]:
```
minmax(A, m, n)
```

```
strategy:  10
```

Out[5]:
```
25
```

In [6]:
```
def maxmin(A, m, n):
    min_ = []
    for i in range(m):
        min_.append(min(A[i]))
    k = -1
    for k in range(m):
        if min_[k] == max(min_):
            b = k
    print("strategy: ", b)
    return max(min_)
```

Минимальная цена игры:

In [7]:
```
maxmin(A, m, n)
```

```
strategy:  14
```

Out[7]:
```
-2
```

3 пункт. Выяснить есть ли решение в чистых стратегиях.

In [8]:
```
if maxmin(A, m, n) == minmax(A, m, n):
    print("Есть решение в чистых стратегиях")
else:
    print("Нет решения в чистых стратегиях")
```

```
strategy:  14
strategy:  10
Нет решения в чистых стратегиях
```

4 пункт. Найти равновесие в смешанных стратегиях и цену игры.

Используем встроенные в модуль PuLP решатель задач линейного программирования.

In [9]:
```
def direct_lp(mat):
    model = LpProblem("Matrix Game", LpMaximize)
    variables = LpVariable.matrix("X", [str(i+1) for i in range(mat.shape[1])],
                                  cat="Continuous", lowBound=0)
    np_vars = np.array(variables)
    model += lpSum(np_vars) # target function
    for i in range(mat.shape[0]):
        model += lpSum(mat[i,:] * np_vars) <= 1, f"Constraint {i+1}"
    model.solve(PULP_CBC_CMD(msg=0))
    status=LpStatus[model.status]
    variables = {key.name: key.value() for key in model.variables()}
    return [v for k, v in sorted(variables.items(),
                                 key=lambda item:int(item[0].split("_")[-1]))]
```

In [10]:
```
A = np.array(A)
```

In [38]:
```
direct_lp(A)
```

Out[38]:
```
[0.0058479532,
 0.0058479532,
 0.0058479532,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298]
```

In [39]:
```
def dual_lp(mat):
    model = LpProblem("Matrix Game", LpMinimize)
    variables = LpVariable.matrix("Y", [str(i+1) for i in range(mat.shape[0])],
                                  cat="Continuous", lowBound=0)
    np_vars = np.array(variables)
    model += lpSum(np_vars) # target function
    for i in range(mat.shape[1]):
        model += lpSum(mat[:,i] * np_vars) >= 1, f"Constraint {i+1}"
    model.solve(PULP_CBC_CMD(msg=0))
    status=LpStatus[model.status]
    variables = {key.name: key.value() for key in model.variables()}
    return [v for k, v in sorted(variables.items(),
                                 key=lambda item:int(item[0].split("_")[-1]))]
```

In [40]:
```
dual_lp(A)
```

Out[40]:
```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0058479532,
 0.0058479532,
 0.0058479532,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0087719298,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]
```

In [14]:
```
def game_cost(mat):
    return sum(dual_lp(mat))**(-1)
```

Цена игры равняется:

In [15]:
```
game_cost(A)
```

Out[15]:
```
11.40000003192
```

In [16]:
```
def opt_strat_first(mat):
    return game_cost(mat) * np.array(dual_lp(mat))

def opt_strat_second(mat):
    return game_cost(mat) * np.array(direct_lp(mat))
```

Находим оптимальные стратегии для первого и второго игрока.

In [17]:
```
print("first strat:", opt_strat_first(A))
print("second strat:", opt_strat_second(A))
```

```
first strat: [0.         0.         0.         0.         0.06666667 0.06666667
 0.06666667 0.1        0.1        0.1        0.1        0.1
 0.1        0.1        0.1        0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
second strat: [0.06666667 0.06666667 0.06666667 0.1        0.1        0.1
 0.1        0.1        0.1        0.1        0.1        ]
```

5 пункт. Повторить пункт 2-4 для матрицы -A и A^T * A

Сделаем преобразование матрицы:

In [18]:
```
def f1(A, m, n):
    from copy import deepcopy
    B = deepcopy(A)
    for i in range(m):
        for j in range(n):
            B[i][j] *= -1
    return B
```

In [19]:
```
B = np.array(f1(A, m, n))
B
```

Out[19]:
```
array([[-21,  -2, -13, -24,  -5, -16,   3,  -8, -19,   0, -11],
       [ -2, -13, -24,  -5, -16,   3,  -8, -19,   0, -11, -22],
       [-13, -24,  -5, -16,   3,  -8, -19,   0, -11, -22,  -3],
       [-24,  -5, -16,   3,  -8, -19,   0, -11, -22,  -3, -14],
       [ -5, -16,   3,  -8, -19,   0, -11, -22,  -3, -14, -25],
       [-16,   3,  -8, -19,   0, -11, -22,  -3, -14, -25,  -6],
       [  3,  -8, -19,   0, -11, -22,  -3, -14, -25,  -6, -17],
       [ -8, -19,   0, -11, -22,  -3, -14, -25,  -6, -17,   2],
       [-19,   0, -11, -22,  -3, -14, -25,  -6, -17,   2,  -9],
       [  0, -11, -22,  -3, -14, -25,  -6, -17,   2,  -9, -20],
       [-11, -22,  -3, -14, -25,  -6, -17,   2,  -9, -20,  -1],
       [-22,  -3, -14, -25,  -6, -17,   2,  -9, -20,  -1, -12],
       [ -3, -14, -25,  -6, -17,   2,  -9, -20,  -1, -12, -23],
       [-14, -25,  -6, -17,   2,  -9, -20,  -1, -12, -23,  -4],
       [-25,  -6, -17,   2,  -9, -20,  -1, -12, -23,  -4, -15],
       [ -6, -17,   2,  -9, -20,  -1, -12, -23,  -4, -15,   4],
       [-17,   2,  -9, -20,  -1, -12, -23,  -4, -15,   4,  -7],
       [  2,  -9, -20,  -1, -12, -23,  -4, -15,   4,  -7, -18],
       [ -9, -20,  -1, -12, -23,  -4, -15,   4,  -7, -18,   1],
       [-20,  -1, -12, -23,  -4, -15,   4,  -7, -18,   1, -10],
       [ -1, -12, -23,  -4, -15,   4,  -7, -18,   1, -10, -21],
       [-12, -23,  -4, -15,   4,  -7, -18,   1, -10, -21,  -2],
       [-23,  -4, -15,   4,  -7, -18,   1, -10, -21,  -2, -13],
       [ -4, -15,   4,  -7, -18,   1, -10, -21,  -2, -13, -24],
       [-15,   4,  -7, -18,   1, -10, -21,  -2, -13, -24,  -5],
       [  4,  -7, -18,   1, -10, -21,  -2, -13, -24,  -5, -16],
       [ -7, -18,   1, -10, -21,  -2, -13, -24,  -5, -16,   3],
       [-18,   1, -10, -21,  -2, -13, -24,  -5, -16,   3,  -8],
       [  1, -10, -21,  -2, -13, -24,  -5, -16,   3,  -8, -19]])
```

Максимальная цена игры для матрицы -A

In [20]:
```
minmax(B, m, n)
```

```
strategy:  10
```

Out[20]:
```
4
```

Минимальная цена игры для матрицы -A

In [21]:
```
maxmin(B, m, n)
```

```
strategy:  22
```

Out[21]:
```
-23
```

In [22]:
```
if maxmin(B, m, n) == minmax(B, m, n):
    print("Есть решение в чистых стратегиях")
else:
    print("Нет решения в чистых стратегиях")
```

```
strategy:  22
strategy:  10
Нет решения в чистых стратегиях
```

In [23]:
```
direct_lp(B)
```

Out[23]:
```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

In [24]:
```
dual_lp(B)
```

Out[24]:
```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.25,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]
```

In [25]:
```
game_cost(B)
```

Out[25]:
```
4.0
```

Вот тут решатель не смог найти оптимальную стратегию второго игрока, и цену игры он считает неадекватно, я не могу понять в чем может быть проблема.

In [26]:
```
print("first strat:", opt_strat_first(B))
print("second strat:", opt_strat_second(B))
```

```
first strat: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
second strat: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Сделаем второе преобразование над матрицей:

In [27]:
```
def transpose(A):
    return [[row[i] for row in A] for i in range(len(A[0]))]
```

In [28]:
```
def prois(U, L, D=None):
    if (len(U[0]) == len(L)):
        R = [[sum([U[i][k] * L[k][j] for k in range(len(U[i]))]) for j in range(len(L[0]))] for i in range(len
        if D != None:
            return prois(R,D)
        return R
    else:
        print('err in Mult')
        exit()
```

In [29]:
```
def f2(A, m, n):
    from copy import deepcopy
    B = deepcopy(A)
    return prois(transpose(B), B)
```

In [30]:
```
C = f2(A, m, n)
C = np.array(C)
C
```

Out[30]:
```
array([[5455, 2210, 2895, 4210, 1955, 3630, 3235, 2270, 4635, 2530, 2555],
       [2210, 5114, 2378, 2642, 3836, 2090, 3344, 3458, 2072, 4316, 2720],
       [2895, 2378, 5551, 2394, 2867, 4330, 2163, 3686, 3379, 2202, 4715],
       [4210, 2642, 2394, 5386, 2108, 2850, 4132, 2234, 3576, 3148, 2240],
       [1955, 3836, 2867, 2108, 4979, 2300, 2531, 4412, 2003, 3224, 3395],
       [3630, 2090, 4330, 2850, 2300, 5530, 2340, 2930, 4300, 2100, 3680],
       [3235, 3344, 2163, 4132, 2531, 2340, 5299, 2468, 2787, 4036, 2195],
       [2270, 3458, 3686, 2234, 4412, 2930, 2468, 5546, 2444, 2972, 4340],
       [4635, 2072, 3379, 3576, 2003, 4300, 2787, 2444, 5491, 2268, 2915],
       [2530, 4316, 2202, 3148, 3224, 2100, 4036, 2972, 2268, 5194, 2420],
       [2555, 2720, 4715, 2240, 3395, 3680, 2195, 4340, 2915, 2420, 5555]])
```

Верхняя цена игры при использовании чистой стратегии.

In [31]:
```
minmax(C, C.shape[0], C.shape[1])
```

```
strategy:  4
```

Out[31]:
```
4979
```

Нижняя цена игры при использовании чистой стратегии вторым игроком.

In [32]:
```
maxmin(C, C.shape[0], C.shape[1])
```

```
strategy:  7
```

Out[32]:
```
2234
```

In [33]:
```
if maxmin(C, C.shape[0], C.shape[1]) == minmax(C, C.shape[0], C.shape[1]):
    print("Есть решение в чистых стратегиях")
else:
    print("Нет решения в чистых стратегиях")
```

```
strategy:  7
strategy:  4
Нет решения в чистых стратегиях
```

In [34]:
```
direct_lp(C)
```

Out[34]:
```
[2.6113004e-05,
 3.3188785e-05,
 4.1780806e-05,
 3.1335604e-05,
 6.6040628e-05,
 3.1335604e-05,
 3.1335604e-05,
 0.0,
 2.6113004e-05,
 2.6113004e-05,
 0.0]
```

In [35]:
```
dual_lp(C)
```

Out[35]:
```
[2.6113004e-05,
 2.6113004e-05,
 7.817054e-06,
 3.1335604e-05,
 0.0,
 3.1335604e-05,
 3.1335604e-05,
 0.00010708016,
 2.6113004e-05,
 2.6113004e-05,
 0.0]
```

In [36]:
```
game_cost(C)
```

Out[36]:
```
3191.2580769704764
```

Находим цену игры и умножая на векторы каждого игрока получаем оптимальное решение в смешанных стратегиях.

In [37]:
```
print("first strat:", opt_strat_first(C))
print("second strat:", opt_strat_second(C))
```

```
first strat: [0.08333333 0.08333333 0.02494624 0.1        0.         0.1
 0.1        0.34172043 0.08333333 0.08333333 0.        ]
second strat: [0.08333333 0.10591398 0.13333333 0.1        0.21075269 0.1
 0.1        0.         0.08333333 0.08333333 0.        ]
```

In [ ]: