**Министерство науки и высшего образования Российской Федерации**

**Московский государственный технический университет имени**

**Н. Э. Баумана**

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

**Лабораторная работа №2**

# ПО ДИСЦИПЛИНЕ «ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ»

«Выпукло-вогнутые антагонистические игры»

**Вариант 11**

**Студент**: Кириченков А.А., ИУ8-104
**Преподаватель:** Коннова Н. С.

Москва
2022

# Цель и задачи

**Цель работы –** найти оптимальные стратегии непрерывной выпукло-вогнутой антагонистической игры аналитическим и численными методами.

1 Постановка задачи

Пусть функция выигрыша (ядро) антагонистической игры, заданной на единичном квадрате, непрерывна:

$$H(x,y) \in C(\Pi), \Pi = [0,1] \times [0,1].$$

Тогда существуют нижняя и верхняя цена игры, и, кроме того,

$$h = \overline{h} \equiv \max_{\mathrm{F}} \min_{y} E(F,y) = \min_{G} \max_{x} E(x,G) \equiv \underline{h},$$

а для среднего выигрыша игры имеют место равенства

$$E(x,G) = \int_0^1 H(x,y)\, dG(y), E(F,y) = \int_0^1 H(x,y)\, dF(x),$$

где $F(x)$, $G(y)$ – произвольные вероятностные меры выбора стратегий для обоих игроков, заданные на единичном интервале.

Выпукло-вогнутая игра всегда разрешима в чистых стратегиях.

# Выполнение лабораторной работы

## 1 Аналитическое решение

Функция ядра имеет вид:

$$H(x,y) = -5x^2 + \frac{5}{6}y^2 + \frac{10}{3}xy - \frac{2}{3}x - 2y.$$

Условия принадлежности игры к классу выпукло-вогнутых выполняются:

$$H_{xx} = 2a = -10 < 0,$$

$$H_{yy} = 2b = \frac{10}{6} > 0;$$

Для нахождения оптимальных стратегий найдём производные функции ядра по каждой переменной:

$$H_x = 2ax + cy + d = -10 * \text{x} + \frac{10}{3} * \text{y} - \frac{2}{3},$$

$$H_y = 2by + cx + e = \frac{5}{3} * \text{y} + \frac{10}{3} * \text{x} - 2.$$

При $H_x = 0$ и $H_y = 0$ получим:

$$x = -\frac{cy + d}{2a} = \frac{2}{3}\text{y} - \frac{1}{30},$$

$$y = -\frac{cx + e}{2b} = -2\text{x} + \frac{6}{5}.$$

Поскольку $x \geq 0$ и $y \geq 0$, для максимальных стратегий имеем:

$$\psi(y) = \begin{cases} \frac{2}{3}\text{y} - \frac{1}{30}, y \geq \frac{1}{20}, \\ 0, \quad y \leq \frac{1}{20}; \end{cases}$$

$$\phi(x) = \begin{cases} -2\text{x} + \frac{6}{5}, x \geq \frac{3}{5}, \\ 0, x \leq \frac{3}{5}. \end{cases}$$

Решив систему для $\psi(y)$ и $\phi(x)$ относительно переменных $x$ и $y$, получаем:

$$x^* = \frac{1}{5}, y^* = \frac{4}{5}.$$

При этом седловая точка игры $H(x^*, y^*) = -0.867$.

## 2 Численное решение

Рассмотрим метод аппроксимации функции выигрышей на сетке. При помощи программы (см. Приложение А) найдены решения при различном шаге сетки. В таблице ниже приведены этапы расчёта:

Таблица 1 – Шаги расчёта стратегий методом аппроксимации функции выигрышей на сетке

```
➜ python lab2.py
Load conditions from file? (Y/N): y
Enter filename: data.txt
N = 1
   0.000   -1.167
  -5.667   -3.500
Has saddle point
x = 0, y = 1, h = -1.167
N = 2
   0.000   -0.792   -1.167
  -1.583   -1.542   -1.083
  -5.667   -4.792   -3.500
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 0, y = 1, h = = -1.129
N = 3
   0.000   -0.574   -0.963   -1.167
  -0.778   -0.981   -1.000   -0.833
  -2.667   -2.500   -2.148   -1.611
  -5.667   -5.130   -4.407   -3.500
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 1/3, y = 2/3, h = = -0.983
N = 4
   0.000   -0.448   -0.792   -1.031   -1.167
  -0.479   -0.719   -0.854   -0.885   -0.812
  -1.583   -1.615   -1.542   -1.365   -1.083
  -3.312   -3.135   -2.854   -2.469   -1.979
  -5.667   -5.281   -4.792   -4.198   -3.500
Has saddle point
x = 1/4, y = 3/4, h = -0.885
N = 5
   0.000   -0.367   -0.667   -0.900   -1.067   -1.167
  -0.333   -0.567   -0.733   -0.833   -0.867   -0.833
  -1.067   -1.167   -1.200   -1.167   -1.067   -0.900
  -2.200   -2.167   -2.067   -1.900   -1.667   -1.367
  -3.733   -3.567   -3.333   -3.033   -2.667   -2.233
  -5.667   -5.367   -5.000   -4.567   -4.067   -3.500
Has saddle point
x = 1/5, y = 4/5, h = -0.867
N = 6
   0.000   -0.310   -0.574   -0.792   -0.963   -1.088   -1.167
  -0.250   -0.468   -0.639   -0.764   -0.843   -0.875   -0.861
  -0.778   -0.903   -0.981   -1.014   -1.000   -0.940   -0.833
  -1.583   -1.616   -1.602   -1.542   -1.435   -1.282   -1.083
  -2.667   -2.606   -2.500   -2.347   -2.148   -1.903   -1.611
  -4.028   -3.875   -3.676   -3.431   -3.139   -2.801   -2.417
  -5.667   -5.421   -5.130   -4.792   -4.407   -3.977   -3.500
Has saddle point
x = 1/6, y = 5/6, h = -0.875
N = 7
   0.000   -0.269   -0.503   -0.704   -0.871   -1.003   -1.102   -1.167
  -0.197   -0.398   -0.565   -0.697   -0.796   -0.861   -0.891   -0.888
  -0.599   -0.731   -0.830   -0.895   -0.925   -0.922   -0.884   -0.813
  -1.204   -1.269   -1.299   -1.296   -1.259   -1.187   -1.082   -0.942
```

```
  -2.014   -2.010   -1.973   -1.901   -1.796   -1.656   -1.483   -1.276
  -3.027   -2.956   -2.850   -2.711   -2.537   -2.330   -2.088   -1.813
  -4.245   -4.105   -3.932   -3.724   -3.483   -3.207   -2.898   -2.554
  -5.667   -5.459   -5.218   -4.942   -4.633   -4.289   -3.912   -3.500
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 1/7, y = 6/7, h = = -0.888
N = 8
   0.000   -0.237   -0.448   -0.633   -0.792   -0.924   -1.031   -1.112   -1.167
  -0.161   -0.346   -0.505   -0.638   -0.745   -0.826   -0.880   -0.909   -0.911
  -0.479   -0.612   -0.719   -0.799   -0.854   -0.883   -0.885   -0.862   -0.812
  -0.953   -1.034   -1.089   -1.117   -1.120   -1.096   -1.047   -0.971   -0.870
  -1.583   -1.612   -1.615   -1.591   -1.542   -1.466   -1.365   -1.237   -1.083
  -2.370   -2.346   -2.297   -2.221   -2.120   -1.992   -1.839   -1.659   -1.453
  -3.312   -3.237   -3.135   -3.008   -2.854   -2.674   -2.469   -2.237   -1.979
  -4.411   -4.284   -4.130   -3.951   -3.745   -3.513   -3.255   -2.971   -2.661
  -5.667   -5.487   -5.281   -5.049   -4.792   -4.508   -4.198   -3.862   -3.500
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 1/4, y = 3/4, h = = -0.883
N = 9
   0.000   -0.212   -0.403   -0.574   -0.724   -0.854   -0.963   -1.051   -1.119   -1.167
  -0.136   -0.307   -0.457   -0.586   -0.695   -0.784   -0.852   -0.899   -0.926   -0.932
  -0.395   -0.525   -0.634   -0.722   -0.790   -0.837   -0.864   -0.870   -0.856   -0.821
  -0.778   -0.866   -0.934   -0.981   -1.008   -1.014   -1.000   -0.965   -0.909   -0.833
  -1.284   -1.331   -1.358   -1.364   -1.350   -1.315   -1.259   -1.183   -1.086   -0.969
  -1.914   -1.920   -1.905   -1.870   -1.815   -1.739   -1.642   -1.525   -1.387   -1.228
  -2.667   -2.632   -2.576   -2.500   -2.403   -2.286   -2.148   -1.990   -1.811   -1.611
  -3.543   -3.467   -3.370   -3.253   -3.115   -2.957   -2.778   -2.578   -2.358   -2.117
  -4.543   -4.426   -4.288   -4.130   -3.951   -3.751   -3.531   -3.290   -3.029   -2.747
  -5.667   -5.508   -5.329   -5.130   -4.909   -4.669   -4.407   -4.126   -3.823   -3.500
Has saddle point
x = 2/9, y = 7/9, h = -0.870
N = 10
   0.000   -0.192   -0.367   -0.525   -0.667   -0.792   -0.900   -0.992   -1.067   -1.125   -1.167
  -0.117   -0.275   -0.417   -0.542   -0.650   -0.742   -0.817   -0.875   -0.917   -0.942   -0.950
  -0.333   -0.458   -0.567   -0.658   -0.733   -0.792   -0.833   -0.858   -0.867   -0.858   -0.833
  -0.650   -0.742   -0.817   -0.875   -0.917   -0.942   -0.950   -0.942   -0.917   -0.875   -0.817
  -1.067   -1.125   -1.167   -1.192   -1.200   -1.192   -1.167   -1.125   -1.067   -0.992   -0.900
  -1.583   -1.608   -1.617   -1.608   -1.583   -1.542   -1.483   -1.408   -1.317   -1.208   -1.083
  -2.200   -2.192   -2.167   -2.125   -2.067   -1.992   -1.900   -1.792   -1.667   -1.525   -1.367
  -2.917   -2.875   -2.817   -2.742   -2.650   -2.542   -2.417   -2.275   -2.117   -1.942   -1.750
  -3.733   -3.658   -3.567   -3.458   -3.333   -3.192   -3.033   -2.858   -2.667   -2.458   -2.233
  -4.650   -4.542   -4.417   -4.275   -4.117   -3.942   -3.750   -3.542   -3.317   -3.075   -2.817
  -5.667   -5.525   -5.367   -5.192   -5.000   -4.792   -4.567   -4.325   -4.067   -3.792   -3.500
Has saddle point
x = 1/5, y = 4/5, h = -0.867
N = 11
Has saddle point
x = 2/11, y = 9/11, h = -0.869
N = 12
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 1/6, y = 5/6, h = = -0.874
N = 13
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 3/13, y = 10/13, h = = -0.873
N = 14
Has saddle point
x = 3/14, y = 11/14, h = -0.868
N = 15
Has saddle point
x = 1/5, y = 4/5, h = -0.867
N = 16
Has saddle point
x = 3/16, y = 13/16, h = -0.868
N = 17
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 3/17, y = 14/17, h = = -0.870
N = 18
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 2/9, y = 7/9, h = = -0.870
N = 19
```

```
Has saddle point
x = 4/19, y = 15/19, h = -0.867
N = 20
Has saddle point
x = 1/5, y = 4/5, h = -0.867
N = 21
Has saddle point
x = 4/21, y = 17/21, h = -0.867
N = 22
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 2/11, y = 9/11, h = = -0.869
N = 23
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 5/23, y = 18/23, h = = -0.869
N = 24
Has saddle point
x = 5/24, y = 19/24, h = -0.867
N = 25
Has saddle point
x = 1/5, y = 4/5, h = -0.867
N = 26
Has saddle point
x = 5/26, y = 21/26, h = -0.867
N = 27
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 5/27, y = 22/27, h = = -0.868
N = 28
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 3/14, y = 11/14, h = = -0.868
N = 29
Has saddle point
x = 6/29, y = 23/29, h = -0.867
N = 30
Has saddle point
x = 1/5, y = 4/5, h = -0.867
N = 31
Has saddle point
x = 6/31, y = 25/31, h = -0.867
N = 32
Hasn't saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001
x = 3/16, y = 13/16, h = = -0.868
Found solution on 33 iteration:
x = 0.202, y = 0.798, h = -0.867
Analytical method
Derivate of H =  -5*x**2 + 10*x*y/3 - 2*x/3 + 5*y**2/6 - 2*y
Derivate Hxx =  -10
Derivate Hyy =  5/3
The game is convex-concave
Derivate Hx =  -10*x + 10*y/3 - 2/3
Derivate Hy =  10*x/3 + 5*y/3 - 2
x = 1/5, y = 4/5, h = -0.867
```

Итоговое численное решение (с точностью $\varepsilon < 0.001$):

$$x^* \approx 0.202, \qquad y^* \approx 0.798, \qquad H(x^*, y^*) \approx -0.867.$$

Погрешность между аналитическим и приближенным решением методом аппроксимации на сетке составляет 0%.

# Вывод

В результате выполнения лабораторной работы получены следующие результаты:

- изучен и реализован аналитический и численный метод аппроксимации на сетке нахождения оптимальных стратегий в непрерывной выпукло-вогнутой антагонистической игре двух лиц;
- найдена оптимальная стратегия обоих игроков аналитическим методом: $x^* = 0.2, y^* = 0.8$; седловая точка при этом , $H(x^*, y^*) = -0.867$;
- найдено приближенное решение методом аппроксимации на сетке $x^* \approx 0.202,\ y^* \approx 0.798,\ H(x^*, y^*) \approx -0.867$ с точностью $\varepsilon < 0.001$;
- результаты, полученные аналитическим и численным методом, получились идентичные.

# Приложение А

```
import math
import numpy as np
import fractions
from sympy import Symbol
import warnings
warnings.filterwarnings('ignore')

def get_row_by_index(matrix, index):
    return matrix[index]

def get_column_by_index(matrix, index):
    return [matrix[i][index] for i in range(len(matrix))]

def vector_addition(a, b):
    return [i + j for i, j in zip(a, b)]

def brown_robinson_method(matrix, eps):
    m = len(matrix)
    n = len(matrix[0])

    x = m * [0]
    y = n * [0]

    curr_strategy_a = 0
    curr_strategy_b = 0

    win_a = m * [0]
    loss_b = n * [0]
    curr_eps = math.inf
    k = 0

    lower_bounds = []
    upper_bounds = []

    while (curr_eps > eps):
        k += 1
        win_a = vector_addition(win_a, get_column_by_index(matrix,
curr_strategy_b))
        loss_b = vector_addition(loss_b, get_row_by_index(matrix,
curr_strategy_a))
        x[curr_strategy_a] += 1
        y[curr_strategy_b] += 1

        lower_bound = min(loss_b) / k
        upper_bound = max(win_a) / k
        lower_bounds.append(lower_bound)
        upper_bounds.append(upper_bound)

        curr_eps = min(upper_bounds) - max(lower_bounds)

        curr_strategy_a = np.argmax(win_a)
        curr_strategy_b = np.argmin(loss_b)

    cost = max(lower_bounds) + curr_eps / 2
```

```
    x = [i / k for i in x]
    y = [i / k for i in y]

    return x, y, cost

def kernel_function(x, y, a, b, c, d, e):
    return a * x**2 + b * y**2 + c * x * y + d * x + e * y

def find_saddle_point(mat):
    max_loss = np.amax(mat, axis=0)
    min_max = np.amin(max_loss)
    y = np.argmin(max_loss)

    min_win = np.amin(mat, axis=1)
    max_min = np.amax(min_win)
    x = np.argmax(min_win)

    return max_min if max_min == min_max else 0, x, y

def average(a):
    return sum(a) / len(a)

def limit(a, eps):
    N = -1
    ff = False
    for i in range(0, len(a) - 1):
        ff = True
        for j in range(i + 1, len(a)):
            if abs(a[j] - a[i]) >= eps:
                ff = False
                break
        if ff:
            N = i
            break
    if not ff:
        return math.inf
    return average([min(a[N + 1: ]), max(a[N + 1: ])])

def generate_grid_approximation(n, a, b, c, d, e):
    return [[kernel_function(i / n, j / n, a, b, c, d, e) for j in range(n +
1)] for i in range(n + 1)]

def grid_approximation_method(eps, a, b, c, d, e):
    cost_array = []
    x_array = []
    y_array = []
    n = 1
    while True:
        cur_H, x, y, h, saddle_point = approximation_method_step(eps, n, a, b,
c, d, e)
        cost_array.append(h)
        lim = limit(cost_array, eps)
        if lim != math.inf:
            x_array.append(x)
            y_array.append(y)

        stop_lim = limit(cost_array, fractions.Fraction(eps, 10))
        if stop_lim != math.inf:
```

```
            print(f"Found solution on {n} iteration:")
            print("x = {:.3f}, y = {:.3f}, h =
{:.3f}".format(float(average(x_array)), float(average(y_array)), float(lim)))
            return average(x_array), average(y_array), lim

        print_result(cur_H, n, x, y, h, saddle_point, eps)
        n += 1

def approximation_method_step(eps, n, a, b, c, d, e):
    cur_H = generate_grid_approximation(n, a, b, c, d, e)

    saddle_point, x, y = find_saddle_point(np.asarray(cur_H))
    if saddle_point:
        h = saddle_point
        x = fractions.Fraction(x, n)
        y = fractions.Fraction(y, n)
    else:
        x, y, h = brown_robinson_method(cur_H, eps)
        x = fractions.Fraction(np.argmax(x), n)
        y = fractions.Fraction(np.argmax(y), n)
    return cur_H, x, y, h, saddle_point

def print_result(H,n,x, y, h, saddle_point, eps):
    print(f"N = {n}")
    if n <= 10:
        for i in H:
            print(*["{:8.3f}".format(float(j)) for j in i])

    if saddle_point:
        print("Has saddle point\nx = {:}, y = {:}, h = {:.3f}".format(x, y,
float(saddle_point)))
    else:
        print("Hasn't saddle point")
        print("Calculated with Brown-Robinson method with accuracy eps =
{:.3f}\nx = {:}, y = {:}, h = {:.3f}".format(float(eps), x, y, float(h)))

def ask_user():
    check = str(input("Load conditions from file? (Y/N): ")).lower().strip()
    try:
        if check[0] == 'y':
            return True
        elif check[0] == 'n':
            return False
        else:
            print('Invalid Input')
            return ask_user()
    except Exception as error:
        print("Please enter valid inputs")
        print(error)
        return ask_user()

def get_conditions_file():
    filename = str(input("Enter filename: "))
    lines = []

    with open(filename, 'r') as file:
        lines = file.readlines()
    try:
```

```
        a = fractions.Fraction(lines[0])
        b = fractions.Fraction(lines[1])
        c = fractions.Fraction(lines[2])
        d = fractions.Fraction(lines[3])
        e = fractions.Fraction(lines[4])
    except ValueError as err:
        print(f"Incorrect values: {err}")
    return a, b, c, d, e

def get_condiditions_user_input():
    try:
        a = fractions.Fraction(input("a >> "))
        b = fractions.Fraction(input("b >> "))
        c = fractions.Fraction(input("c >> "))
        d = fractions.Fraction(input("d >> "))
        e = fractions.Fraction(input("e >> "))
    except ValueError as err:
        print(f"Incorrect values: {err}")
    return a, b, c, d, e

def get_conditions():
    return get_conditions_file() if ask_user() else
get_condiditions_user_input()

def analytical_method(a, b, c, d, e):
    x = Symbol('x')
    y = Symbol('y')

    _H = a * x ** 2 + b * y ** 2 + c * x * y + d * x + e * y

    print('Derivate of H = ', _H)
    Hxx = _H.diff(x, 2)
    Hyy = _H.diff(y, 2)
    print('Derivate Hxx = ', Hxx)
    print('Derivate Hyy = ', Hyy)

    if float(Hxx) < 0 and float(Hyy) > 0:
        print("The game is convex-concave")
    else:
        print("The game isn't convex-concave")

    Hx = _H.diff(x)
    Hy = _H.diff(y)

    print('Derivate Hx = ', Hx)
    print('Derivate Hy = ', Hy)

    y_sol = (c * d - 2 * a * e) / (4 * b * a - c * c)
    x_sol = -(c * y_sol + d) / (2 * a)
    h = kernel_function(float(x_sol), float(y_sol), a, b, c, d, e)
    print("x = {:}, y = {:}, h = {:.3f}".format(x_sol, y_sol, h))

def main():
    p = 3
    a, b, c, d, e = get_conditions()
    grid_approximation_method(fractions.Fraction(1, 10**p), a, b, c, d, e)
    print("Analytical method")
    analytical_method(a, b, c, d, e)
```

```
if __name__ == "__main__":
    main()
```