**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»**
**(МГТУ им. Н. Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

# Лабораторная работа №2 на тему "Аналитический и численный методы решения непрерывной выпукло-вогнутой игры"

по дисциплине «Теория игр и исследование операций»

Вариант 12

| | | | |
|---|---|---|---|
| Студент | ИУ8-104<br>(Группа) | Мильченко И. Д.<br>(И. О. Фамилия) | ______<br>(Подпись, дата) |
| Преподаватель | | Коннова Н. С.<br>(И. О. Фамилия) | ______<br>(Подпись, дата) |

*Москва, 2025 г.*

# ЦЕЛЬ РАБОТЫ

Найти оптимальные стратегии непрерывной выпукло-вогнутой антагонистической игры аналитическим и численным методами.

## Задание

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

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

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

$$\overline{h} = \max_x \min_y H(x, y), \quad \underline{h} = \min_y \max_x H(x, y).$$

Для среднего выигрыша игры имеют место равенства:

$$\overline{h} = \int_P H(x, y)\, dF(x)\, dG(y), \quad \underline{h} = \int_P H(x, y)\, dF(x)\, dG(y),$$

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

Выпукло-ковогнутая игра всегда разрешима в смешанных стратегиях.

Пусть функция выигрыша игры имеет вид:

$$H(x, y) = ax^2 + by^2 + cx + dy + e.$$

Исходные данные для выполнения лабораторной работы приведены в таблице 1.

Таблица 1 – Вариант задания

| **Номер варианта** | **a** | **b** | **c** | **d** | **e** |
|---|---|---|---|---|---|
| 12 | -10 | $\frac{40}{3}$ | 40 | -16 | -32 |

# ХОД РАБОТЫ

Для решения данной лабораторной работы был использован язык программирования Go. В проекте был написан класс для решения выпукло-вогнутых игр разными способами (аналитическим и численным соответственно).

Пример запуска программы:

```
go run cmd/lw2/main.go
```

Перейдем к аналитическому методу решения данной игры.

# 1 Аналитический метод

Вывод программы решения игры аналитическим методом представлена на рисунке 1.

```
Kernel function:
-10.00x^2 + 13.33y^2 + 40.00xy + -16.00x + -32.00y

Let us check the feasibility of the conditions forthe game to belong to the convex-concave cla
ss:
H_xx = 2*a = 2 * -10.00 = -20.00 < 0
H_yy = 2*b = 2 * 13.33 = 26.67 > 0
The game presented is convex-concave.

H_x = 2ax + cy + d = -20.00 * x + 40.00 * y + -16.00
H_y = 2by + cx + e = 26.67 * y + 40.00 * x + -32.00

x = [cy + d]/[-2a] = 200.00y + -80.00
y = [cx + e]/[-2b] = -266.67x + 213.33

200.00y + -80.00, if y >= 0.40
0 else
-266.67x + 213.33, if x <= 0.80
0 else

x = 0.60
y = 0.40
H = -12.80
```

Рисунок 1 – Решение выпукло-вогнутой игры аналитическим методом

Функция выигрыша имеет вид:

$$H(x,y) = -10x^2 + \frac{40}{3}y^2 + 40xy - 16x - 32y.$$

При этом

$$\frac{\partial^2 H}{\partial x^2} = -20 < 0, \quad \frac{\partial^2 H}{\partial y^2} = \frac{80}{3} > 0,$$

то есть функция ковогнута по $x$ и выпукла по $y$, что подтверждает принадлежность игры к классу выпукло-ковогнутых.

Равновесная точка $(x, y)$ определяется решением системы условий первого порядка:

$$\frac{\partial H}{\partial x} = 0, \quad \frac{\partial H}{\partial y} = 0.$$

Путём решения системы были найдены оптимальные стратегии игроков:

$$x = 0{,}6, \quad y = 0{,}4,$$

при которых значение функции выигрыша (цена игры) составляет:

$$H(x, y) = -12{,}8.$$

## 2 Численный метод

Для численного решения игры с непрерывной функцией выигрыша используется метод дискретизации ядра на равномерной сетке. На каждой итерации увеличивается шаг сетки, что позволяет уточнять приближённое значение седловой точки.

Введём параметр разбиения $N$ и для всех $N = 1, 2, \ldots$ зададим аппроксимацию функции ядра на единичном квадрате. Для этого разобьём отрезок $[0, 1]$ на $N$ равных частей с шагом $\Delta = \frac{1}{N}$. Тогда узлы сетки будут иметь координаты:

$$x_i = \frac{i}{N}, \quad y_j = \frac{j}{N}, \quad i, j = 0, \ldots, N.$$

На этой сетке формируется матрица приближённых значений функции выигрыша:

$$H^{(N)} = \left(H_{ij}^{(N)}\right), \quad \text{где} \quad H_{ij}^{(N)} = H\left(\frac{i}{N}, \frac{j}{N}\right), \quad i, j = 0, \ldots, N.$$

Если на шаге с номером $N$ седловая точка не обнаружена, применяется метод Брауна–Робинсон для численного поиска решения соответствующей матричной игры.

Алгоритм завершает работу, когда разность между оценками цены игры на последних $k$ итерациях (по умолчанию $k = 5$) становится меньше заданного порога точности $\varepsilon = 0.01$. При увеличении количества итераций рекомендуется также увеличивать параметр $k$, чтобы обеспечить стабильность сходимости.

Вывод программы алгоритма для первых 8 итераций представлен на рисунке 2.

```
N = 2
[     0   -12.7   -18.7 ]
[-10.5   -13.2   -9.17 ]
[   -26   -18.7   -4.67 ]
Brown-Robinson solution:
x = 0.500 y = 0.500 H = -13.167

N = 3
[     0   -9.19   -15.4   -18.7 ]
[-6.44   -11.2    -13   -11.8 ]
```

```
[-15.1  -15.4  -12.7  -7.11 ]
[  -26  -21.9  -14.7  -4.67 ]
Brown-Robinson solution:
x = 0.333 y = 0.667 H = -12.963

N = 4
[    0  -7.17  -12.7  -16.5  -18.7 ]
[-4.62  -9.29  -12.3  -13.6  -13.3 ]
[-10.5  -12.7  -13.2    -12  -9.17 ]
[-17.6  -17.3  -15.3  -11.6  -6.29 ]
[  -26  -23.2  -18.7  -12.5  -4.67 ]
Brown-Robinson solution:
x = 0.500 y = 0.500 H = -13.167

N = 5
[    0  -5.87  -10.7  -14.4  -17.1  -18.7 ]
[ -3.6  -7.87  -11.1  -13.2  -14.3  -14.3 ]
[   -8  -10.7  -12.3  -12.8  -12.3  -10.7 ]
[-13.2  -14.3  -14.3  -13.2  -11.1  -7.87 ]
[-19.2  -18.7  -17.1  -14.4  -10.7  -5.87 ]
[  -26  -23.9  -20.7  -16.4  -11.1  -4.67 ]
Saddle point found:
x = 0.400 y = 0.600 H = -12.800

N = 6
[    0  -4.96  -9.19  -12.7  -15.4  -17.4  -18.7 ]
[-2.94  -6.80  -9.91  -12.3  -13.9  -14.8  -14.9 ]
[-6.44  -9.19  -11.2  -12.4    -13  -12.7  -11.8 ]
[-10.5  -12.1    -13  -13.2  -12.6  -11.2  -9.17 ]
[-15.1  -15.6  -15.4  -14.4  -12.7  -10.3  -7.11 ]
[-20.3  -19.7  -18.4  -16.3  -13.5  -9.91  -5.61 ]
[  -26  -24.3  -21.9  -18.7  -14.7  -10.1  -4.67 ]
Brown-Robinson solution:
x = 0.333 y = 0.667 H = -12.963

N = 7
[    0  -4.30  -8.05  -11.3  -13.9  -16.1  -17.6  -18.7 ]
[-2.49  -5.97  -8.91  -11.3  -13.2  -14.5  -15.2  -15.4 ]
[-5.39  -8.05  -10.2  -11.8  -12.8  -13.3  -13.2  -12.6 ]
[-8.69  -10.5  -11.9  -12.6  -12.8  -12.5  -11.6  -10.2 ]
[-12.4  -13.4  -13.9  -13.9  -13.3  -12.1  -10.4  -8.22 ]
[-16.5  -16.7  -16.4  -15.6  -14.1  -12.2  -9.67  -6.63 ]
```

```
[-21.1  -20.5  -19.3  -17.6  -15.4  -12.6  -9.31  -5.44 ]
[  -26  -24.6  -22.6  -20.1  -17.1  -13.5  -9.35  -4.67 ]
Brown-Robinson solution:
x = 0.429 y = 0.571 H = -12.830

N = 8
[    0  -3.79  -7.17  -10.1  -12.7  -14.8  -16.5  -17.8  -18.7 ]
[-2.16  -5.32  -8.07  -10.4  -12.3  -13.8  -14.9  -15.6  -15.8 ]
[-4.62  -7.17  -9.29   -11  -12.3  -13.2  -13.6  -13.7  -13.3 ]
[-7.41  -9.32  -10.8  -11.9  -12.6  -12.8  -12.7  -12.1  -11.1 ]
[-10.5  -11.8  -12.7  -13.1  -13.2  -12.8    -12  -10.8  -9.17 ]
[-13.9  -14.6  -14.8  -14.7  -14.1  -13.1  -11.7  -9.82  -7.57 ]
[-17.6  -17.7  -17.3  -16.5  -15.3  -13.7  -11.6  -9.17  -6.29 ]
[-21.7  -21.1  -20.1  -18.7  -16.8  -14.6  -11.9  -8.82  -5.32 ]
[  -26  -24.8  -23.2  -21.1  -18.7  -15.8  -12.5  -8.79  -4.67 ]
Brown-Robinson solution:
x = 0.375 y = 0.625 H = -12.823
```

Рисунок 2 – Результаты работы численного алгоритма при разных значениях $N$

Были найдены оптимальные стратегии игроков за 42 итерации:

$$x = 0{,}59, \quad y = 0{,}41,$$

при которых значение функции выигрыша (цена игры) составляет:

$$H(x^*, y^*) = -12{,}8.$$

## 3 Сравнительная оценка погрешностей

Полученные результаты и их приведены в сводной таблице 2. Таким образом полученное приближенное решение удовлетворяет заданной точности $\varepsilon \leq 0.1$.

Таблица 2 – Сводная таблица сравнительной оценки погрешностей

| | $H$ | $x$ | $y$ |
|---|---|---|---|
| Аналитический метод | -12.8 | 0.6 | 0.4 |
| Численный метод | -12.8 | 0.59 | 0.41 |
| Абсолютная погрешность, $\Delta$ | 0 | 0.1 | 0.1 |
| Относительная погрешность, % | 0 | 1.67 | 2.5 |

# ЗАКЛЮЧЕНИЕ

В данной работе была исследована непрерывная антагонистическая выпукло-ковогнутая игра двух лиц, а также аналитический и численный методы её решения.

Сначала было получено точное эталонное решение с помощью аналитического метода:

$$x = 0{,}6; \quad y = 0{,}4; \quad H = -12{,}8.$$

Затем был применён численный метод на основе аппроксимации функции выигрыша и использования метода Брауна–Робинсон. При заданной точности $\varepsilon \leq 0{,}01$ было получено приближённое решение:

$$x \approx 0{,}59; \quad y \approx 0{,}41; \quad H \approx -12{,}8.$$

В заключение была проведена сравнительная оценка погрешностей. Полученные численные значения хорошо согласуются с аналитическими, а отклонения по координатам $x$ и $y$ не превышают допустимый уровень. Таким образом, численный метод показал высокую эффективность и применимость для приближённого решения непрерывных выпукло-ковогнутых игр.

# ПРИЛОЖЕНИЕ А

## Класс решения выгнуто-вогнутых игр

Листинг А.1 – game.go

```
package convexconcane

import (
  "errors"
  "fmt"
  "math"
  "slices"
  "strings"

  gamematrix "github.com/themilchenko/game_theory/internal/
     game_matrix"
  brownrobinson "github.com/themilchenko/game_theory/internal/
     game_matrix/brown_robinson"
)

const (
  kernelFunc = "%.2fx^2 + %.2fy^2 + %.2fxy + %.2fx + %.2fy\n"
  eps         = 0.01
  lastIters  = 4
)

type ConvexConcane struct {
  a float64
  b float64
  c float64
  d float64
  e float64
}

func New(a, b, c, d, e float64) (*ConvexConcane, error) {
  if 2*a >= 0 || 2*b <= 0 {
    return nil, errors.New("game is not convex-concane")
  }

  return &ConvexConcane{
    a: a,
    b: b,
```

```
    c: c,
    d: d,
    e: e,
  }, nil
}

func (c *ConvexConcane) h(x, y float64) float64 {
  return c.a*x*x + c.b*y*y + c.c*x*y + c.d*x + c.e*y
}

func (c *ConvexConcane) SolveAnalytical() *Solution {
  sol := &Solution{
    strB: &strings.Builder{},
  }

  fmt.Fprintln(sol.strB, "Kernel function:")
  fmt.Fprintf(sol.strB, kernelFunc, c.a, c.b, c.c, c.d, c.e)

  fmt.Fprintln(sol.strB)

  fmt.Fprintln(sol.strB, "Let us check the feasibility of the
     conditions for"+
    "the game to belong to the convex-concave class:")
  fmt.Fprintf(sol.strB, "H_xx = 2*a = 2 * %.2f = %.2f < 0\n", c.a,
     2*c.a)
  fmt.Fprintf(sol.strB, "H_yy = 2*b = 2 * %.2f = %.2f > 0\n", c.b,
     2*c.b)
  fmt.Fprintln(sol.strB, "The game presented is convex-concave.")

  fmt.Fprintln(sol.strB)

  fmt.Fprintf(sol.strB, "H_x = 2ax + cy + d = %.2f * x + %.2f * y +
      %.2f\n", 2*c.a, c.c, c.d)
  fmt.Fprintf(sol.strB, "H_y = 2by + cx + e = %.2f * y + %.2f * x +
      %.2f\n", 2*c.b, c.c, c.e)

  fmt.Fprintln(sol.strB)

  x := fmt.Sprintf("%.2fy + %.2f", c.c/-2*c.a, c.d/-2*c.a)
  y := fmt.Sprintf("%.2fx + %.2f", c.c/-2*c.b, c.e/-2*c.b)

  fmt.Fprintf(sol.strB, "x = [cy + d]/[-2a] = %s\n", x)
```

```
  fmt.Fprintf(sol.strB, "y = [cx + e]/[-2b] = %s\n", y)

  fmt.Fprintln(sol.strB)

  fmt.Fprintf(sol.strB, "%s, if y >= %.2f\n0 else\n", x, -c.d/c.c)
  fmt.Fprintf(sol.strB, "%s, if x <= %.2f\n0 else\n", y, -c.e/c.c)

  fmt.Fprintln(sol.strB)

  sol.Y = (c.e - (c.c*c.d)/(2*c.a)) / ((c.c*c.c)/(2*c.a) - 2*c.b)
  sol.X = -1 * ((c.c*sol.Y + c.d) / (2 * c.a))
  sol.H = c.h(sol.X, sol.Y)

  return sol
}

func (c *ConvexConcane) SolveNumerical() *Solution {
  sol := &Solution{
    strB: &strings.Builder{},
  }

  N := 2

  lastNIters := make([]float64, 0, lastIters)
  var gameCost, prevGameCost float64
  var xEstimate, yEstimate float64

  for !c.isFinish(lastNIters) {
    m := c.makeMatrix(N)

    fmt.Fprintf(sol.strB, "N = %d\n", N)
    fmt.Fprintln(sol.strB, m.MatrixString())

    highestPrice, highestIdx := m.HighestPrice()
    lowestPrice, lowestIdx := m.LowestPrice()

    if highestPrice != lowestPrice {
      fmt.Fprintln(sol.strB, "Brown-Robinson solution:")

      s := m.SolveBrownRobinson(brownrobinson.Epsilon(0.01))
```

```
      xEstimate = float64(slices.Index(s.X, slices.Max(s.X))) /
         float64(N)
      yEstimate = float64(slices.Index(s.Y, slices.Max(s.Y))) /
         float64(N)
      gameCost = c.h(xEstimate, yEstimate)
    } else {
      fmt.Fprintln(sol.strB, "Saddle point found:")

      gameCost = highestPrice
      xEstimate = float64(lowestIdx) / float64(N)
      yEstimate = float64(highestIdx) / float64(N)
    }

    fmt.Fprintf(sol.strB, "x = %.3f y = %.3f H = %.3f\n\n",
      xEstimate, yEstimate, gameCost)

    // prevGameCost will appear only since third iteration.
    if N != 2 {
      lastNIters = c.addBuf(lastNIters, math.Abs(gameCost-
         prevGameCost))
    }

    prevGameCost = gameCost
    N++
  }

  sol.H = gameCost
  sol.X = xEstimate
  sol.Y = yEstimate

  return sol
}

func (c *ConvexConcane) addBuf(last []float64, el float64) []
   float64 {
  if len(last) == lastIters {
    for i := range len(last) - 1 {
      last[i] = last[i+1]
    }
    last[lastIters-1] = el

    return last
```

```
  }

  return append(last, el)
}

func (c *ConvexConcane) isFinish(iters []float64) bool {
  if len(iters) != lastIters {
    return false
  }

  sum := float64(0)

  for _, v := range iters {
    sum += v
  }

  return sum <= eps
}

func (c *ConvexConcane) makeMatrix(N int) *gamematrix.GameMatrix {
  res := make([][]float64, 0, N+1)

  for i := range N + 1 {
    r := make([]float64, 0, N+1)

    for j := range N + 1 {
      r = append(r, c.h(float64(i)/float64(N), float64(j)/float64(N
        )))
    }

    res = append(res, r)
  }

  m, _ := gamematrix.New(res)

  return m
}
```

Листинг А.2 – solution.go

```
package convexconcane
```

```go
import (
  "fmt"
  "strings"
)

type Solution struct {
  X float64
  Y float64
  H float64

  strB *strings.Builder
}

func (s *Solution) String() string {
  fmt.Fprintf(s.strB, "x = %.2f\ny = %.2f\nH = %.2f", s.Y, s.X, s.H
     )

  return s.strB.String()
}
```

## ПРИЛОЖЕНИЕ Б

### Точка входа в программу

Листинг Б.3 – main.go

```go
package main

import (
  "fmt"
  "log"

  convexconcane "github.com/themilchenko/game_theory/internal/
     convex-concane"
)

var (
  a float64 = -10
  b float64 = 40 / float64(3)
  c float64 = 40
  d float64 = -16
  e float64 = -32
)

func main() {
  c, err := convexconcane.New(a, b, c, d, e)
  if err != nil {
    log.Fatal(err)
  }

  fmt.Println(c.SolveAnalytical())
  fmt.Println(c.SolveNumerical())
}
```