МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛТАВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ЮРІЯ КОНДРАТЮКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТА ТЕЛЕКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ, ІНФОРМАТИКИ ТА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ

ДИПЛОМНА РОБОТА СПЕЦІАЛІСТА

зі спеціальності 7.04030201 «Інформатика»

на тему

«Методи розв’язування матричних ігор та їх програмна реалізація»

Студента групи 501-ЕІ Фесюри Сергія Леонідовича

Керівник роботи
кадидат фіз.-мат. наук,
доцент Радченко Г. О.

Завідувач кафедри
доктор фіз.-мат. наук,
професор Губреєв Г.М.

Полтава 2012

РЕФЕРАТ

Дипломна робота спеціаліста: 115 с., 1 малюнок, 1 додаток, 10 джерел.

Об’єкт дослідження: матричні ігри.

Мета роботи: розглянути точні і наближені методи розв’язування матричних ігор, створити програми, що їх реалізують та здійснити візуалізацію створених програм.

Методи: графічний, метод зведення матричних ігор до задач ЛП, метод Брауна-Робінсона. Програма реалізована в середовищі Komodo Edit на мові JavaScript.

Відповідно до поставленого завдання у роботі досліджено відомі раніше підходи до розв’язування матричних ігор. Здійснена програмна реалізація трьох основних методів.

Ключові слова: ТЕОРІЯ ІГОР, МАТРИЧН ІГРИ, СИМПЛЕКС-МЕТОД, МЕТОД БРАУНА-РОБІНСОНА, МАТЕМАТИЧНІ МЕТОДИ В РІЗНИХ ГАЛУЗЯХ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

$U_1=\{\alpha_1, \alpha_2,..., \alpha_m\}$ – множина чистих стратегій першого гравця.

$U_2=\{\beta_1, \beta_2, ... \beta_n\}$ – множина чистих стратегій другого гравця.

$U_1 \times U_2$ — декартовий добуток множин стратегій гравців, множина ситуацій гри.

$$A=\begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_m \end{array}\begin{array}{c} \begin{array}{cccc} \beta_1 & \beta_2 & \dots & \beta_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}$$ - матриця виграшів першого та програшів другого гравця.

$\Gamma = \{U_1, U_2, A\}$ матрична гра двох осіб.

ВСТУП

Організації звичайно мають цілі, які суперечать цілям інших організацій-конкурентів. Тому робота менеджерів часто полягає у виборі рішення з урахуванням дій конкурентів. Для вирішення таких проблем призначені методи теорії ігор.

Теорія ігор - це розділ прикладної математики, який вивчає моделі і методи прийняття оптимальних рішень в умовах конфлікту.

Під конфліктом розуміється така ситуація, в якій зіштовхуються інтереси двох або більше сторін, що переслідують різні (найчастіше суперечливі) цілі. При цьому кожне рішення має прийматися в розрахунку на розумного суперника, який намагається зашкодити іншому учаснику гри досягти успіху.

З метою дослідження конфліктної ситуації будують її формалізовану спрощену модель. Для побудови такої моделі необхідно чітко описати конфлікт, тобто:

1. уточнити кількість учасників (учасники або сторони конфлікту називаються гравцями);
2. вказати на всі можливі способи (правила) дій гравців, які називаються стратегіями гравців;
3. розрахувати, якими будуть результати гри, якщо кожний гравець вибере певну стратегію (тобто з’ясувати виграші або програші гравців).

Основну задачу теорії ігор можна сформулювати так: визначити, яку стратегію має застосувати розумний гравець у конфлікті з розумним суперником, щоб гарантувати кожному з них виграш, при чому відхилення будь-якого з гравців від оптимальної стратегії може тільки зменшити його виграш.

Математична теорія ігор здатна не тільки вказати оптимальний шлях до вирішення деяких проблем, а й прогнозувати їх результат. Матричні ігри серйозно вивчаються фахівцями, так як вони досить прості і до них можуть бути зведені ігри загального виду. Тому теорія матричних ігор добре розвинена, існують різні методи пошуку розв'язку ігор.

Але в більшості випадків розв'язування матричних ігор являє собою важкий і

громіздкий процес.

Крім того, виграші гравців у кожній ситуації не завжди визначаються точними вимірами. В процесі збору даних про досліджуване явище, аналізу цих даних та введення при побудові моделі різних припущень накопичуються помилки. Вони можуть позначатися на елементах матриці виграшів. Тому точність у визначенні значення гри та оптимальних стратегій гравців виправдана не завжди.

Також слід зауважити, що похибка в оцінці гравцем свого виграшу не може привести до практично серйозних наслідків і невелике відхилення гравця від оптимальної стратегії не тягне за собою істотної зміни в його виграші.

Тому виникає потреба в розробці чисельних методів розв'язання матричних ігор. В даний час в теорії ігор відомі кілька способів наближеного розв’язку матричних ігор.

Робота складається зі вступу, трьох розділів і додатку, в якому приведена програму на мові JavaScript, яка дозволяє знаходити наближений розв’язок матричної гри.

У першому розділі наведений інформаційний огляд задач, що зводяться до матричних ігор, а також короткий огляд методів.

Розділ другий присвячений більш глибокому викладу різних методів розв’язання матричних ігор.

У третьому розділі описано реалізацію деяких методів розв’язання ігор з описом архітектурних моментів програми.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Основні поняття і означення теорії матричних ігор

В роботі розглядаються парні антагоністичні ігри, тобто такі, в яких приймають участь тільки два гравці, і виграш одного гравця рівний програшу іншого. Вважається, що кожен гравець має скінченну кількість чистих стратегій (тобто скінченну кількість можливих способів дій). Позначатимемо: $U_1=\{\alpha_1, \alpha_2, ..., \alpha_m\}$ – множина стратегій першого гравця; $U_2=\{\beta_1, \beta_2, ... \beta_n\}$ – множина чистих стратегій другого гравця.

Множина $U_1 \times U_2$ — декартовий добуток множин стратегій гравців називається множиною ситуацій гри. Для кожної ситуації повинна бути визначена ціна гри. Так як гра антагоністична достатньо визначити виграш *a* одного з гравців, наприклад першого. Тоді виграш другого буде рівний (-*a*). Таким чином визначається матриця виграшів першого гравця (для другого гравця матриця виграшів буде -A):

$$A=\begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_m \end{array}\begin{array}{c} \begin{array}{cccc} \beta_1 & \beta_2 & \dots & \beta_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}$$

Визначення. Система Г = {U1, U2, A} називається матричною грою двох осіб. Розігрування матричної гри зводиться до вибору гравцем 1 і-го рядка матриці виграшів, а гравцем 2 - j-го стовпця. Після цього гравець 1 отримує виграш рівний a_{ij}, а гравець 2 - $(-a_{ij})$. При правильній грі гравець 1 може завжди гарантувати собі виграш, який назвемо нижнім значенням ціни гри. Позначимо його: $\underline{v}=\max\limits_i \min\limits_j a_{ij}$. У свою чергу, гравець 2 може гарантувати собі програш, який назвемо верхнім значенням ціни гри. Позначимо його: $\bar{v}=\min\limits_j \max\limits_i a_{ij}$. Чисті стратегії i* і j*, що відповідають $\underline{v}$ і $\bar{v}$ називаються максимінною і мінімаксною стратегіями.

Лемма 1. В матричній грі $\underline{v} \leqslant \bar{v}$.

Визначення. Ситуація (i*, j*) називається ситуацією рівноваги, якщо для i$\in$1, 2, ..., m, j$\in$1, 2, ..., n виконується нерівність:.$a_{ij^*} \leq a_{i^*j^*} \leq a_{i^*j}$ Якщо ситуація рівноваги для гри існує, то така гра називається грою з сідловою точкою, а елемент $a_{i^*j^*}$ матриці A, що їй відповідає називається сідловою точкою цієї матриці. Ситуація рівноваги це така ситуація, від якої жодному з гравців не вигідно відхилятися. В цьому випадку стратегії i*, j* називають оптимальними стратегіями гравців. Щоб така ситуація існувала необхідно і достатньо рівність верхньої та нижньої цін гри, тобто $\underline{v} = \bar{v} = v$.

Визначення. Нехай (i*, j*) - ситуація рівноваги в матричній грі. Тоді число $v = a_{i^*j^*}$ називається значенням або ціною гри. Наприклад, у грі Г з матрицею

$A = \begin{pmatrix} 5 & 6 & 5 \\ 2 & 5 & 1 \\ 1 & 4 & 2 \end{pmatrix}$ існує не одна ситуація рівноваги. В даній грі їх дві: (1, 1) і (1, 3).

Множина всіх ситуацій рівноваги в матричній грі позначимо через Z(Г).

Лемма про масштаб 1. Нехай Г і Г' - дві матричні ігри з матрицею виграшів A = $\{a_{ij}\}$ і A' = $\{a'_{ij}\}$, причому A' = β A + α I, β = const, α = const, I - одинична. Тоді Z(Г) = Z(Г') і v' = β v + α (де v' - значення ціни гри Г', v - значення ціни гри Г).

Ця лема має велике практичне значення, так як більшість алгоритмів для розв'язування матричних ігор засновано на припущенні, що матриця гри позитивна. У випадку, коли матриця має непозитивні елементи, слід додати до всіх елементів матриці число найбільше за абсолютною величиною, з усіх негативних елементів. Існують ігри, в яких ситуації рівноваги в чистих стратегіях не існує. Тоді гравцям буває не вигідно дотримуватися своїх мінімаксних і максимінних стратегій, так як вони можуть отримати більший виграш, відхилившись від них. В цьому випадку гравцям розумно діяти випадково, тобто вибирати стратегії довільно і не повідомляти про вибір суперника. Такі стратегії гравців будемо називати мішаними.

Визначення. Мішаною стратегією гравця називається повний набір ймовірностей застосування його чистих стратегій.

Так якщо гравець 1 має m чистих стратегій, то його мішана стратегія x - це набір чисел x = (x_1, x_2, ..., x_m), які задовольняють співвідношенням $x_i \geq 0$, $\sum_{i=1}^{m} x_i = 1$.

Аналогічним чином визначається мішана стратегія у гравця 2.

Визначення. Оптимальними стратегіями гравців називаються стратегії, які при багаторазовому повторенні забезпечують гравцям максимально можливий середній виграш (або мінімально можливий середній програш). Таким чином, процес гри при використанні гравцями своїх мішаних стратегій перетворюється на випадкове випробування, яке назвемо ситуацією в мішаних стратегіях. Вона позначається так (x, y), де x і y - мішані стратегії гравців 1 і 2 відповідно.

Для ситуації в мішаних стратегіях кожен гравець визначає для себе середній виграш, який виражається у вигляді математичного очікування його виграшів:.K(x, y) = $\sum_{i}^{m}\sum_{j}^{n} a_{ij} x_i y_j$ Від матричної гри прийшли до нової гри $\bar{\Gamma}$ = {X, Y, K}, де X, Y - множини мішаних стратегій гравців, а K - функція виграшів в мішаних стратегіях. Таку гру називають мішаним розширенням матричної гри. Цілі гравців залишаються незмінними: гравець 1 бажає отримати максимальний виграш, а гравець 2 прагне звести свій програш до мінімуму. Тому для мішаного розширення гри, аналогічним чином визначаються верхнє і нижнє значення ціни гри, тільки тепер гравці вибирають свої мішані стратегії. Позначимо їх:

$$\bar{v} = \min_{j} \max_{i} K(x, y) \quad ,$$

$$\underline{v} = \max_{i} \min_{j} K(x, y)$$

В цьому випадку залишається справедливою лема 1, тобто $\underline{v} \leqslant \bar{v}$

Визначення. Ситуація (x*, y*) в грі утворює ситуацію рівноваги, якщо для всіх x ∈ X, y ∈ Y виконується рівність: K (x, y *) ≤ K (x *, y *) ≤ K (x *, y). Щоб ситуація рівноваги в мішаному розширенні гри існувала необхідно і достатньо рівності

верхньої та нижньої цін гри, тобто $\underline{v} = \bar{v} = v$, де v - ціна гри. Для випадку мішаного розширення гри також справедлива лема про масштаб.

Лемма про масштаб 2. Нехай Γ_A і $\Gamma_{A'}$ - дві матричні ігри $A' = \alpha A + B$, $\alpha = \text{const}$, B - матриця з однаковими елементами β, тобто $\beta_{ij} = \beta$ для всіх i, j. Тоді $Z(\bar{\Gamma}_A) = Z(\Gamma_{A'})$ і $v_{A'} = \alpha v_A + \beta$ (де $n_{A'}$ - значення ціни гри $\Gamma_{A'}$, v_A - значення ціни гри Γ_A).

Теорема (фон Неймана). У мішаному розширенні матричної гри завжди існує ситуація рівноваги.

Визначення. Стратегії, яким відповідають однакові значення платіжної матриці (тобто матриця містить однакові рядки(стовпці)), називаються дублюючими. Якщо всі елементи і-го рядка (стовпця) платіжної матриці перевищують значення елементів j-го рядка (стовпця), то кажуть, що і-та стратегія гравця А (гравця В) є домінуючою над j-ю, а j-та називається домінованою і-ю.

1.2 Огляд методів розв’язування антагоністичних ігор

Існує кілька основних методів розв'язання матричних ігор. Розглянемо основні з них. Це метод зведення матричної гри до задач ЛП, графічний метод, ітеративний метод Брауна-Робінсона та монотонний ітеративний алгоритм.

Зведення матричної гри до задач лінійного програмування

Алгоритм пошуку розв’язку матричної антагоністичної гри, заданої платіжною матрицею можна звести до алгоритму симплекс-методу розв'язання пари взаємодвоїстих задач лінійного програмування. Розглянемо метод, за допомогою якого можна привести скінченну матричну антагоністичну гру до двох взаємодвоїстих задач лінійного програмування.

Нехай антагоністична гра задана платіжною матрицею А, що має розмірність m × n, і ця гра не є грою з сідловою точкою. Необхідно знайти розв’язок гри, тобто визначити оптимальні мішані стратегії першого і другого гравців:

$P^*= (p_1^*, p_2^*, \ldots, p_m^*)$, $Q^*= (q_1^*, q_2^*, \ldots, q_n^*)$

де P^* і Q^* - вектори, компоненти яких p_i^* і p_j^* є ймовірностями застосування чистих стратегій і та j відповідно першим і другим гравцями і відповідно для них виконуються співвідношення:

$$p_1^* + p_2^* + \ldots + p_m^* = 1,\ q_1^* + q_2^* + \ldots + q_n^* = 1$$

$$A= \begin{array}{c} \\ p_1^* \\ p_1^* \\ \ldots \\ p_m^* \end{array} \begin{array}{c} \begin{array}{cccc} q_1^* & q_2^* & \ldots & q_n^* \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix} \end{array}$$

Знайдемо спочатку оптимальну стратегію першого гравця P^*. Ця стратегія повинна забезпечити виграш першого гравця не менше V, тобто ≥ V, при будь-якій поведінці другого гравця, і виграш, рівний V, при його оптимальній поведінці, тобто при стратегії Q^*.

Ціна гри V нам поки невідома. Без обмеження загальності, можна припустити

її рівною деякому позитивному числу V> 0. Дійсно, для того, щоб виконувалася умова V> 0, достатньо, щоб всі елементи матриці А були невід'ємними. Цього можна домогтися за допомогою афінних перетворень: додаючи до всіх елементів матриці А одну і ту ж досить велику позитивну константу М; при цьому ціна гри збільшиться на М, а розв'язок не зміниться. Отже, будемо вважати V> 0.

Припустимо, що перший гравець А застосовує свою оптимальну стратегію p^*, а другий гравець В свою чисту стратегію j, тоді середній виграш $\alpha_j = K(p^*_{q_j})$ першого гравця А буде рівним:

$$K(p^*_{q_j}) = a_{1j} p_1^* + a_{2j} p_2^* + \ldots + a_{mj} p_n^*$$

Оптимальна стратегія першого гравця (А) володіє тією властивістю, що при будь-якій поведінці другого гравця (В) забезпечує виграш першого гравця, не менший, ніж ціна гри V; значить, будь-яке з чисел a_j не може бути менше V ($\geq$ V). Отже, при оптимальної стратегії, повинна виконуватися наступна система нерівностей:

$$\begin{cases} a_{11} p_1^* + a_{21} p_2^* + \ldots + a_{m1} p_m^* \geq V \\ a_{12} p_1^* + a_{22} p_2^* + \ldots + a_{m2} p_m^* \geq V \\ \ldots\ldots\ldots\ldots\ldots \\ a_{1n} p_1^* + a_{2n} p_2^* + \ldots + a_{mn} p_m^* \geq V \end{cases} \quad (2.1.1)$$

Розділимо нерівності (2.1.1) на позитивну величину V (праві частини системи (2.1.1)) і введемо позначення:

$$y_1 = \frac{p_1^*}{V}, y_2 = \frac{p_2^*}{V}, \ldots, y_m = \frac{p_m^*}{V} \quad (2.1.2)$$

$$y_1 \geq 0, y_2 \geq 0, .., y_m \geq 0, \quad (2.1.3)$$

Тоді умови (2.1.1) запишуться в вигляді:

$$\begin{cases} a_{11} y_1 + a_{21} y_2 + \ldots + a_{m1} y_m \geq 1 \\ a_{12} y_1 + a_{22} y_2 + \ldots + a_{m2} y_m \geq 1 \\ \ldots\ldots\ldots\ldots\ldots \\ a_{1n} y_1 + a_{2n} y_2 + \ldots + a_{mn} y_m \geq 1 \end{cases} \quad (2.1.4)$$

де $y_1, y_2, \ldots y_m$ – невід'ємні змінні. В силу (2.1.2) и того, що $p_1^* + p_2^* + \ldots + p_m^* = 1$

змінні $y_1, y_2, ..., y_m$ задовольняють умову, яку позначимо через F:

$$F = y_1 + y_2 + ... + y_m = \frac{1}{V} \quad (2.1.5)$$

Оскільки перший гравець свій гарантовани виграш (V) старається зробити максимально можливим (V → max), очевидно що при цьому права частина (2.1.5) - $\frac{1}{V}$ →min приймає мінімальне значення. Таким чином задача розв'язування антагоністичної гри для першого гравця звелась до наступної математичної задачі:

Визначити такі невід'ємні значення змінних $y_1, y_2, ... y_m$, щоб вони задовольняли систему лінійних обмежень в вигляді нерівностей(2.1.4), системі загальних обмежень (2.1.3) и мінімізували цільову функцію F:

$$F = y_1 + y_2 + _{...} + y_m \rightarrow \min \quad (2.1.6)$$

Це типова задача лінійного програмування і вона може бути розв'язана симплекс-методом. Таким чином, розв'язуючи задачу лінійного програмування, ми можемо знайти оптимальну стратегію $P^* = (p_1^*, p_2^*, ..., p_m^*)$ гравця А.

Знайдемо тепер оптимальну стратегію $Q^* = (q_1^*, q_2^*, ..., q_n^*)$ гравця В. Все буде аналогічно розв'язуванню гри для гравця А, з тією різницею, що гравець В прагне не максимізувати, а мінімізувати виграш (по суті справи його програш), а значить, не мінімізувати, а максимізувати величину $\frac{1}{V}$, Тому що V → min. Замість умов (2.1.4) повинні виконуватися умови:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leqslant 1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leqslant 1 \\ \quad\quad ..... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leqslant 1 \end{cases} \quad (2.1.7)$$

де

$$x_1 = \frac{q_1^*}{V}, x_2 = \frac{q_2^*}{V}, ..., x_n = \frac{q_m^*}{V} \quad (2.1.8)$$

$$x_1 \geq 0, x_2 \geq 0, .., x_n \geq 0 \quad (2.1.9)$$

Потрібно так вибрати змінні $x_1, x_2, .., x_n,$ щоб вони задовольняли умовам (2.1.7), (2.1.9) і приводили в максимум лінійну функцію мети $F^{/}$:

$$F' = x_1 + x_2 + x_n = \frac{1}{V} \to max$$

Таким чином, задача знаходження оптимальної стратегії другого гравця звелася до наступного математичної задачі:

Визначити невід'ємні значення змінних $x_1, x_2, .., x_n,$ щоб вони задовольняли системі лінійних обмежень у вигляді нерівностей (2.1.7), системі загальних обмежень (2.1.8) і максимізувати цільову функцію $F^{/}$:

$$F' = x_1 + x_2 + .. + x_n \to \min \quad (2.1.10)$$

Очевидно, що ця задача є двоїста до задачі (2.1.3)-(2.1.5).

Отже, для того, щоб розв’язати матричну гру треба розв’язати пару двоїстих задач (2.1.3),(2.1.4), (2.1.6) та (2.1.7), (2.1.9), (2.1.10).

Це типова задача лінійного програмування (пряма) і вона може бути вирішена симплекс - методом. Розв’язавши цю пару двоїстих задач знайдемо $x^* = (x_1^*, x_2^*, ..., x_m^*)$ і $y^* = (y_1^*, y_2^*, ..., y_n^*)$, а тоді за формулами $V = \frac{1}{\sum_{i=1}^{m} x_i}$ ($V = \frac{1}{\sum_{j=1}^{n} y_i}$) $q_j^* = V_{xi}$, $i = \overline{1, m}$, $p_i^* = V_{yj}$, $j = \overline{1, n}$ знайдемо ціну гри та координати оптимальної стратегії.

Ітеративний метод Брауна-Робінсона

Часто в практичних задачач немає необхідності знаходити точний розв’язок матричної гри. Досить знайти наближений розв’язок, який дає середній виграш, близький до ціни гри і наближені оптимальні стратегії гравців.

Орієнтовне значення ціни гри може дати вже простий аналіз матриці виграшів та визначення нижньої і верхньої ціни гри. Якщо вони близькі, то пошуками точного розв’язку займатися не обов'язково, так як досить вибрати чисті мінімаксну та максимінну стратегії. Якщо ж вони не близькі, можна отримати прийнятний для практики розв’язок за допомогою чисельних методів розв'язання ігор, за допомогою

одного з чисельних методів, наприклад, за допомогою методу ітерацій. Метод полягає в наступному.

Нехай розігрується матрична гра Γ_A з матрицею $A = \{a_{ij}\}$ розміру (m×n). Ідея методу - багаторазове фіктивне розігрування гри із заданою матрицею. Одне розігрування гри будемо називати партією, число яких необмежено.

У 1-й партії обидва гравці вибирають абсолютно довільні чисті стратегії. Нехай гравець 1 вибрав і-ю стратегію, а гравець 2 - j-у стратегію. У другій партії гравець 1 відповідає на хід гравця 2 тієї своєю стратегією, яка дає йому максимальний сумарний виграш. У свою чергу, гравець 2, відповідає на цей хід гравця 1 своєю стратегією, яка звертає його сумарний програш у мінімум. Далі третя партія.

З ростом числа кроків процесу мішані стратегії, які приписуються гравцям, наближаються до їх оптимальних стратегій. Цей процес наближеного знаходження оптимальних стратегій гравців називається *ітеративним,* а його кроки - ітераціями.

Детальний опис розв’язування матричних ігор за допомогою ітеративного методу Брауна-Робінсона буде наведено в частині 2 даної роботи.

Монотонний ітеративний алгоритм розв'язання матричних ігор

Цей алгоритм реалізується тільки для одного гравця на відміну від методу Брауна-Робінсона, який працює для двох гравців. Алгоритм дозволяє знаходити точно і наближено оптимальну стратегію гравця 1 і значення ціни гри n. За допомогою цього алгоритму можна отримати задану точність розв’язку, причому число кроків, необхідних для досягнення результатів, слабо залежить від розмірності матриці виграшів.

Особливість цього алгоритму у здатності генерувати строго монотонно зростаючу послідовність оцінок ціни гри, що не властиво алгоритму Брауна-Робінсона.

Розглянемо мішане розширення $\bar{\Gamma}_A = (X, Y, K)$ матричної гри Γ_A з матрицею A розміру (m×n). Процес розігрування гри складається з декількох кроків. Нехай кожен з гравців має скінченне число чистих стратегій.

Введемо наступні позначення:

a_i - і-й рядок матриці виграшів;

$x^N=(\xi_1^N,\xi_2^N,\ldots,\xi_m^N) \in X$ - m-мірний вектор, наближення оптимальної стратегії першого гравця на N-кроці (N-номер кроку);

$c^N=(\gamma_1^N,\gamma_2^N,\ldots,\gamma_n^N)$ - n-мірний вектор, який визначає середній накопичений виграш на N-кроці.

Задамо початкові умови. Нехай на 0-кроці $c^0 = a_{i_0}$, $X^0 = (0, \ldots, 1, \ldots, 0)$, де 1 займає i_0-у позицію.

Визначимо ітеративний процес наступним чином: за відомим векторах x^{N-1}, c^{N-1} знаходимо вектори x^N і c^N, які обчислюються за такими формулами:

$$x^N=(1-\varepsilon_N)x^{N-1}+\varepsilon_N\tilde{x}^N;$$
$$c^N=(1-\varepsilon_N)c^{N-1}+\varepsilon_N\tilde{c}^N;$$

де параметр $0 \le \varepsilon_N \le 1$, а вектори $\tilde{x}^N, \tilde{c}^N$ вводяться далі.

Як зазначалося, вектор c^N визначає середній накопичений виграш гравця 1 на N кроці. Компоненти цього вектора - це числа. У гіршому випадку гравець 1 може отримати мінімальне з цих чисел. Приймемо його за нижню оцінку ціну гри, яку позначимо: $\underline{\nu}^{N-1} = \min\limits_{j=1,\ldots,n} \gamma_j^{N-1}$

Запам'ятаємо множину індексів $J^{N-1}=(j_1^{N-1},\ldots,j_k^{N-1})$, (k<n), на яких буде досягається цей мінімум, тобто $\min\limits_{j=1,\ldots,n} \gamma_j^{N-1} = \gamma_{j_1}^{N-1} = \gamma_{j_2}^{N-1} = \ldots = \gamma_{j_k}^{N-1}$

Далі розглянемо підгру Γ^N гри Γ_A з матрицею виграшів $A^N=\{a_{ij^{N-1}}\}$, i=1,…,m, $j^{N-1} \in J^{N-1}$. Матриця виграшів складається із стовпців даної матриці, номери яких визначаються множиною індексів J^{N-1}. У цій підгрі Γ^N знаходимо одну з оптимальних мішаних стратегій гравця 1: $\tilde{x}^N=(\tilde{\xi}_1^N,\ldots,\xi_m^N)$.

Після знаходження $\tilde{x}^N$, Знаходимо вектор $\tilde{c}^N=(\tilde{\gamma}_1^N,\ldots,\tilde{\gamma}_n^N)$ за правилом: $\tilde{c}^N = \sum\limits_{i=1}^{m} \tilde{\xi}_i^N a_i$

І розглянемо гру (2*n), в якій у гравця 1 дві чисті стратегії, а у гравця 2 - n чистих стратегій. Ця гра задається матрицею $\begin{bmatrix} \gamma_1^{N-1} & \dots & \gamma_n^{N-1} \\ \widetilde{\gamma}_1^{N} & \dots & \widetilde{\gamma}_n^{N-1} \end{bmatrix}$, розв'язуючи яку, знаходимо ймовірність використання гравцем 1 своєї стратегії. Це дає нам коефіцієнт ε_N.

Далі обчислюємо x^N, c^N і переходимо до наступного кроку. Процес продовжуємо до тих пір, поки не виконається рівність $\varepsilon_N = 0$, тому що по теоремі про мінімакс $\underline{\nu}^N \le \nu$, а їх рівність (що й потрібно) досягається в цьому випадку, або поки не буде досягнута необхідна точність обчислень.

Збіжність алгоритму гарантується теоремою.

Теорема. Нехай $\{x^N\}$, $\{v^N\}$ - послідовності, що визначаються рівностями (3), (4). Тоді справедливі наступні твердження:

1. $\underline{\nu}^{N-1} < \underline{\nu}^N$ тобто послідовність $\{v^{N-1}\}$ строго монотонно зростає.
2. $\lim_{N\to\infty} \underline{\nu}^N = \nu$
3. $\lim_{N\to\infty} x^N = x^*$, де $x^* \in X^*$ - оптимальна стратегія гравця 1.

Доведення цієї теореми досить рутинно. Його можна подивитися в [9].

1.3 Огляд задач, що зводяться до антагоністичної гри

При розв’язанні економічних задач, у тому числі й маркетингових, часто доводиться аналізувати ситуації, за яких стикаються інтереси двох або більше конкуруючих сторін, переслідуючих різні цілі, особливо це характерне для ринкової економіки. Такого роду ситуації називаються конфліктними. Математичною теорією розв’язання конфліктних ситуацій є теорія ігор. У грі можуть стикатися інтереси двох (гра парна) або декількох (гра множинна) супротивників; існує гра з нескінченною множиною гравців. Якщо у множинній грі гравці утворять коаліції, то гра називається коаліційною; якщо таких коаліцій дві, то гра зводиться до парної.

На промислових підприємствах теорія ігор може використовуватися для вибору оптимальних рішень, наприклад, при створенні раціональних запасів сировини, матеріалів, напівфабрикатів, коли протидіють дві тенденції: збільшення запасів, що гарантують безперебійну роботу виробництва, і скорочення запасів з метою мінімізації витрат на зберігання їх. Розв’язання подібних задач вимагає повної визначеності в формулюванні їх умов (правил гри): встановлення кількості гравців, можливих виграшів (програші розуміють як від’ємний виграш). Кількість стратегій у кожного гравця може бути скінченною і нескінченною, залежно від цього ігри поділяють на скінченні та нескінченні.

Важливими поняттями є поняття оптимальної стратегії, ціни гри, середнього виграшу. Ціна гри V дорівнює математичному сподіванню M виграшу першого гравця, якщо обидва гравці виберуть оптимальні для себе стратегії P* і Q*:

$$V = M (P^*, Q^*).$$

Одним з основних видів ігор є матричні ігри, які називаються парними іграми з нульовою сумою (тобто один гравець виграє стільки, скільки програє другий), за умови, що кожний гравець має скінченну кількість стратегій. У багатьох ігрових задачах у сфері економіки, а також у сфері маркетингу, невизначеність випливає не через свідому протидію супротивника, а через недостатню обізнаність щодо умов, в яких діють сторони, тобто коли невідомі стратегії сторін. Тоді до розгляду додається ще матриця ризиків. Для розв’язання таких задач використовуються критерії

Лапласа, Вальда, Гурвіца та ін.

На основі методів розв'язування статистичних ігор можна сформулювати підходи до розв'язування різноманітних прикладних економічних задач.

Зведення економічних колізій до ігрових задач

Перші два етапи творчої складової процесу прийняття рішення утворюють основу ігрової моделі. Відомо багато прикладів успішного застосування ігрової моделі як у сфері виробничої діяльності, так і на макроекономічному рівні. У прикладах, що наводитимуться далі, обмежимося постановкою та інтерпретацією розв'язків ігрових задач. Наведемо деякі приклади ігрового моделювання в економіці, а також покажемо розширені можливості зведення економічних ситуацій до задач теорії ігор.

Дилема ув'язненого та олігопольні ринки

Олігопо́лія (англ. *Oligopoly*) - структура ринку, при якій в одній галузі домінує невелика кількість конкуруючих фірм, при цьому хоча б одна або дві з них, виробляють значну долю продукції даної галузі, а поява нових продавців ускладнена чи неможлива. Товар, реалізований олігополістичними фірмами, може бути як диференційованим так і стандартизованим.

Два ув'язнених очікують рішення суду за спільно вчинене злодіяння. Запобігши можливості змови, їм висунули умови: якщо зізнаються обидва, то кожен отримає по п'ять років тюрми; якщо зізнається один, то він отримає лише один рік, а другий — 10 років; якщо ж обидва не зізнаються, то кожен отримає по два роки ув'язнення.

Побудову ігрової моделі почнемо з формулювання множин рішень для кожного з ув'язнених. Обидва вони мають для вибору дві взаємовиключні чисті стратегії: перша — зізнатися (s_1 чи Θ_1), друга — не зізнатися (s_2 чи Θ_2). Ефективність кожної з чистих стратегій для кожного з гравців відобразимо відповідно у вигляді функціоналів оцінювання:

$F' = (f'_{kj} : k = 1, 2; j = 1, 2);$

$F'' = (f''_{kj} : k = 1, 2; j = 1, 2),$

де f'_{kj} та f''_{kj} – плата першого та другого ув'язненого відповідно, якщо перший гравець вибрав свою k-ту чисту стратегію S_k (k = 1, 2), а другий – свою j – ту чисту стратегію θ_j (j = 1, 2).

Числовими еквівалентами платежів є терміни ув'язнення гравців, що беруться з протилежним знаком. У цьому випадку матриці платежів мають позитивну складову:

$$F' = F'^{+} = \begin{array}{c} \\ S_1 \\ S_2 \end{array} \begin{array}{c} \begin{array}{cc} \theta_1 & \theta_2 \end{array} \\ \begin{pmatrix} -5 & -1 \\ -10 & -2 \end{pmatrix} \end{array}; \quad F'' = F''^{+} = \begin{array}{c} \\ S_1 \\ S_2 \end{array} \begin{array}{c} \begin{array}{cc} \theta_1 & \theta_2 \end{array} \\ \begin{pmatrix} -5 & -10 \\ -1 & -2 \end{pmatrix} \end{array}.$$

Отримана гра не є антагоністичною, тому що гравцям вигідніше домовитися між собою. З точки зору економічної теорії дилема ув'язненого пояснює, чому продавці на олігопольному ринку намагаються досягти домовленості замість конкуренції, оскільки остання була би вигіднішою для покупців. У випадку, коли на ринку функціонує невелика кількість фірм-продавців однорідної продукції, ціни характеризуються жорсткістю: жодна з фірм не може ні довіряти іншим, ні очікувати, що її конкурент призначить нижчу ціну.

Ситуація, коли на ринку функціонує невелика кількість фірм, носить назву конкуренції серед не багатьох: випадок, коли є декілька продавців продукції, носить назву олігополії, а коли декілька покупців певного виду витрат — назву олігопсонії. Визначальною властивістю конкуренції серед не багатьох є те, що всі конкуруючі фірми можуть впливати на ціни продукції або витрати і при цьому прибуток кожної фірми залежить від стратегії всіх конкуруючих фірм. Слід відмітити важливу спільну рису між конкуренцією серед не багатьох і теорією ігор. В обох випадках результат (прибуток чи виграш) для одного учасника (фірми чи гравця) залежить від діяльності (витрат чи стратегій) решти учасників.

Гра у “старі” та “нові” товари

Нехай у нашому розпорядженні є три види “старих” товарів s_1, s_2, s_3, які надходять на ринок уже давно і попит на які добре відомий. З певного моменту в торговельну мережу починають надходити “нові” товари θ_1, θ_2, θ_3, які можуть замінити “старі”. Тобто “нові” товари зменшують попит на “старі” товари. З попередніх обстежень попиту відомі ймовірності продажу “старих” товарів у разі появи в торговельній мережі “нових” товарів. Для гри у “старі” та “нові” товари платіжна матриця записана у табл. 1.1. Оскільки максимін рівний мінімаксу і рівний 0,7, то гра має сідлову точку, створювану мінімаксними стратегіями $s_{ko} = s_2$ та $\Theta_{jo} = \Theta_2$. Ці стратегії є стійкими у тому розумінні, що відхилення від них невигідне для обох гравців.

Таблиця 1.1

Старі товари	Нові товари		
	Θ_1	Θ_2	Θ_3
s_1	0,5	0,6	0,8
s_2	0,9	0,7	0,8
s_3	0,7	0,5	0,6

4.4 Інвестування капіталу

Інвестор взяв у борг гроші під 1,5% з метою інвестування цих засобів в акції різних компаній. Наявні два види акцій, норми прибутку яких є випадковими величинами і залежать від станів економічного середовища (випадкових обставин). На ринку можуть мати місце тільки дві ситуації: перша (Θ_1) з імовірністю q = 0,2 і друга (Θ_2)— з імовірністю q = 0,8.

Акції реагують на ці ситуації (стани економічного середовища) по-різному: курс акцій першого виду (s_1) у першій ситуації зростає на 5%, а в другій — на 1,25%; курс акцій другого виду (s_2) у першій ситуації падає на 1%, а в другій — зростає на 2,75%.

Необхідно найкращим чином розподілити наявний капітал між цими активами. Попередньо проаналізуємо ці акції з позиції таких їх характеристик, як сподівана

норма прибутку (математичне сподівання норми прибутку) та величина ризику (дисперсія норми прибутку). Обчислимо ці характеристики:

$$m_1 = \sum_{j=1}^{2} (q_j r_{1j}) = 5 \times 0{,}2 + 1{,}25 \times 0{,}8 = 2;$$

$$m_2 = \sum_{k=1}^{2} (q_j r_{2j}) = -1 \times 0{,}2 + 2{,}75 \times 0{,}8 = 2;$$

$$\sigma_1^2 = \sum_{j=1}^{2} (q_j r_{1j}^2) - m_1^2 = 5^2 \times 0{,}2 + (1{,}25)^2 \times 0{,}8 - 2^2 = 2{,}25;$$

$$\sigma_2^2 = \sum_{j=1}^{2} r_{2j}^2 - m_2^2 = (-1)^2 \times 0{,}2 + (2{,}75)^2 \times 0{,}8 - 2^2 = 2{,}25.$$

Хоча значення сподіваних норм прибутку і ризиків збіглися ($m_1 = m_2 = 2$; $\sigma_{21} = \sigma_{22} = 2{,}25$), у випадку інвестування всього капіталу в акції одного виду перевагу слід віддати акціям другого виду. На користь такого вибору свідчать такі міркування. У випадку придбання акцій тільки першого виду банкрутство інвестора може відбутися за настання другої ситуації (1,25 < 1,5), тобто з імовірністю $q_2 = 0{,}8$. Якщо ж придбати акції тільки другого виду, то банкрутство інвестора відбудеться вже у випадку настання першої ситуації (-1 < 1,5,) тобто з імовірністю $q_1 = 0{,}2$. Оскільки $q_1 = 0{,}2 < 0{,}8 = q_2$, то ризик банкрутства в разі інвестування лише в акції другого виду менший за ризик банкрутства в разі інвестування лише в акції першого виду. Але, як бачимо, повністю уникнути банкрутства при інвестуванні всього капіталу в акції другого виду неможливо.

Дослідимо ефект диверсифікації, тобто ефект від розподілу грошових ресурсів між обома активами в найбільш вигідних і безпечних пропорціях. Нехай x_1 — частка капіталу, інвестованого в акції першого виду, тоді $x_2 = 1 - x1$ — частка капіталу, інвестованого в акції другого виду, вектор $X = (x_1; x_2)$ — структура портфеля акцій. Знайдемо характеристики портфеля. Його сподівана норма прибутку

$$m_{\Pi} = m_1 x_1 + m_2 x_2 = 2x_1 + 2(1 - x_1) = 2$$

величина ризику

$$\sigma_{\Pi}^2 = \sigma_1^2 x_1^2 + \sigma_2^2 x_2^2 + 2 p_{12} \sigma_1 \sigma_2 x_1 x_2 = (\sigma_1^2 + \sigma_2^2 + 2 p_{12} \sigma_1 \sigma_2) x_1^2 - 2(\sigma_2^2 - p_{12} \sigma_1 \sigma_2) x_1 + \sigma_2^2$$

де p_{12} – коефіцієнт кореляції між нармами прибутку акцій, обчислюваний за

формулою

$$p_{12}=\frac{\sum_{j=1}^{n}(q_j r_{1j} r_{2j})-m_1 m_2}{\sigma_1 \sigma_2}$$

З урахуванням того, що $\sigma_1 = \sigma_2 = 1{,}5$; $p_{12} = -1$ (тобто має місце абсолютно від'ємна кореляція), отримуємо, що величина ризику портфеля, як функція від частки x1, обчислюється формулою

$$\sigma П_2 = 9x12 - 9x1 + 2{,}25.$$

У даному випадку найкращим портфелем слід уважати портфель з найменшою величиною ризику. Позначимо його структуру через $X = (x_1^*; x_2^*)$. При x_1 функція досягає свого мінімального значення. Згідно з необхідною умовою екстремуму функції для знаходження x_1 слід скористатися рівнянням

$$\frac{d\sigma_{\text{п}}^2}{dx_1}=0 \quad \Rightarrow \quad \left(9x_1^2-9x_1+2{,}25\right)'=0 \quad \Rightarrow \quad 18x_1-9=0 \Rightarrow$$
$$\Rightarrow \quad x_1^*=0{,}5; \quad x_2^*=1-x_1^*=0{,}5 \quad \Rightarrow \quad X^*=(0{,}5;\ 0{,}5).$$

Отриманий результат вказує на те, що розподіл капіталу на рівні частки (по 50%) між акціями обох видів дає змогу, в певному сенсі, позбутися ризику ($\sigma П_2 = 0$). Окрім того, за такого розподілу грошей інвестору не загрожує банкрутство, оскільки для будь-якого стану економічного середовища норма прибутку портфеля, що має структуру X = (0,5; 0,5), становить 2% > 1,5%.

Із задачею побудови оптимального портфеля цінних паперів у даному випадку мають безпосередній зв'язок дві матриці:

а) коваріаційна матриця:

$$C=\begin{pmatrix}\sigma_{11} & \sigma_{12}\\ \sigma_{21} & \sigma_{22}\end{pmatrix}=\begin{pmatrix}2{,}25 & -2{,}25\\ -2{,}25 & 2{,}25\end{pmatrix}$$

де $\sigma_{kj}=\sigma_k \sigma_j \rho_{kj}\,(k=1{,}2\,;\ j=1{,}2);$

б) матриця норм прибутку

$$R=\begin{pmatrix}r_{11} & r_{12}\\ r_{21} & r_{22}\end{pmatrix}=\begin{pmatrix}5 & 1{,}25\\ -1 & 2{,}75\end{pmatrix}$$

де r_k – величина норми прибутку акції k-го виду для j-го стану економічного середовища (k = 1,2; j = 1,2);

Розглянемо парну гру з нульовою сумою, що визначається матрицею С. Легко переконатись у тому, що ця гра не має сідлової точки, а її розв'язком є пара оптимальних мішаних стратегій s_p та q_Q, яким відповідають вектори P = Q = (0,5; 0,5) = X.

При цьому ціна гри V = 0 = $(\sigma П)_2$.

Парна гра з нульовою сумою, що визначається матрицею F, також не має сідлової точки і при цьому оптимальній мішаній стратегії першого гравця (інвестора) відповідає вектор P = (0,5; 0,5) = X. Ціна цієї гри V = 2.

Збіг оптимальних мішаних стратегій, що є розв'язком обох розглянутих ігор, не випадковий.

4.5 Ігрова модель задачі побудови портфеля активів

Лауреат Нобелівської премії Г.Марковіц у своїх дослідженнях вивчав імовірнісну модель ринку активів. У його моделі норма прибутку кожного активу розглядається як випадкова величина

$$R_k = \frac{C_k - C_k^0 + Д_k}{C_k^0} \times 100\%, \qquad k = 1,..., m,$$

де C_k^0 — ціна активу на початок даного періоду C_k– ціна активу (випадкова величина) на кінець даного періоду; $Д_k$– дивіденди (теж випадкова величина), нараховані протягом даного періоду. Конкретне значення норми прибутку k-го активу залежить від стану економічного середовища, тобто визначається ситуацією, що склалася на ринку. Розмірність множини Θ станів економічного середовища може бути довільною, але ми вважатимемо її скінченною і рівною n, тобто Θ = ($Θ_1$;...; $Θ_n$). Кожному стану ринку $Θ_j$ поставимо у відповідність імовірність настання його q_j. Всі ці ймовірності згрупуємо у вектор Q = (q_1;...; q_n). Очевидно, що компоненти вектора Q мають задовольняти співвідношення $\sum_{j=1}^{n} q_j = 1$.

Уважається, що можливі значення норми прибутку k-го активу (k = 1,... m) для всіх можливих станів ринку $Θ_j$ = (1,…n) є відомими і відображаються випадковою величиною R_k = (r_{k1};…; r_{kn}), тобто актив k-го виду приносить r_{k1} одиниць прибутку з

розрахунку на кожну одиницю вкладень, якщо економічне середовище знаходитиметься в своєму j-му стані. Інвестор має можливість інвестувати свої ресурси більш як в один актив, утворити портфель, тобто розподілити свої ресурси між різними активами в найвигіднішій і безпечній пропорції. А тому інвестор хоче оптимізувати структуру портфеля $X = (x_1;\ldots;x_m)$ шляхом визначення оптимальних розмірів часток x_k $(k = 1,\ldots m)$, які відповідають вартості активів кожного виду в загальному обсязі інвестованих у портфель ресурсів.

У своїй моделі Марковіц відштовхується від того, що інвестор для прийняття інвестиційних рішень в якості критеріїв оптимальності використовує лише дві характеристики активів та їх портфелів: сподівану норму прибутку (у формі математичного сподівання) та величину ризику (у формі дисперсії).

Вибір цих кількісних характеристик у якості критеріїв оптимальності дає можливість розглядати задачу побудови портфеля як двокритеріальну. До речі, інвестор схильний вкласти весь свій капітал лише в актив одного виду, якщо цей актив, порівняно з будь-яким портфелем, буде найкращим за обома цими критеріями одночасно, тобто матиме найбільшу сподівану норму прибутку та найменшу величину ризику.

Якщо вважати відомими закони розподілу випадкових величин $R_k(1,\ldots.,m)$, то не становитиме великих труднощів обчислення математичних сподівань $m_k = M(R_k)$ та коваріацій $\sigma_{kj} = cov(R_k; R_j)(k = 1,\ldots..,m; j = 1,\ldots..,m).$

Якщо відома структура портфеля $X = (x_1;\ldots..;x_m)$, то норма прибутку цього портфеля обчислюється за формулою

$$R_{П} = \sum_{k=1}^{m} (x_k R_k)$$

а його характеристики (сподівана норма прибутку портфеля і дисперсія) – за формулою

$$m_{П} = M(R_{П}) = \sum_{k=1}^{m} (x_k) m_k \;\; ; \;\; \sigma_{П}^2 = D(R_{П}) = \sum_{k=1}^{m} \sum_{k=1}^{m} (x_k x_j \sigma_{kj}) \;\; .$$

Ураховуючи зроблені раніше викладки, можна стверджувати, що задача

оптимізації структури портфеля пов'язана з декількома матрицями, кожну з яких можна вибрати в якості функціонала оцінювання статистичної гри. Розглянемо функціонал оцінювання

$$R = R^{+} = (r_{kj}^{+} : l = 1,\dots, m;\ j = 1,\dots, n) = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \dots & \dots & \dots & \dots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{pmatrix}$$

у k-му рядку якого розміщено можливі значення норми прибутку R_k активу k-го виду, а в j-му стовпчику — величини норм прибутків активів усіх видів, що відповідають j-му стану ринку (k = 1,…, m; j = 1,…, n). Мішану стратегію першого гравця, якій відповідає вектор P = (p_1;…; p_m) у грі, що визначається матрицею R, можна тепер інтерпретувати як портфель, а ймовірність p_k — як частку капіталу, інвестованого в актив k-го виду (k = 1,…, m). Розв'язок гри в чистих стратегіях відповідатиме «однорідному» портфелю, тобто портфелю, складеному тільки з активів одного виду. А вибір розв'язку гри у мішаних стратегіях вказує на факт формування портфеля з різних активів. За виконання певних умов оптимальна мішана стратегія з вектором P = (p_1;…; p_m) відповідає ефективному портфелю.

Аналогічна відповідність має місце між оптимальними мішаними стратегіями гравців у парній грі з нульовою сумою, що визначається, з одного боку, коваріаційною матрицею C = (σ_{kj}: k = 1,… m; j = 1,… m), з іншого — ефективним портфелем.

Оптимізація структури портфеля відіграє особливу роль у сучасній економічній теорії та практиці. Можна стверджувати, що будь-яка економічна проблема зводиться до задачі найкращого розподілу ресурсів (матеріальних, фінансових, трудових тощо), наявних у розпорядженні СПР (інвестора) між активами різних видів.

4.6 Формування портфеля інвестиційних проектів

Інвестор вивчає m інвестиційних проектів (їх пронумеровано від 1 до m) щодо вибору серед них найнадійніших з подальшим включенням їх в інвестиційний портфель, враховуючи при цьому свої реальні можливості.

Вибрані найбільш надійні інвестиційні проекти утворюють деяку підмножину множини наявних m проектів. Цю підмножину інтерпретуватимемо як нечітку підмножину всіх інвестиційних проектів. Згідно з означенням нечіткої на елементах множини всіх інвестиційних проектів необхідно визначити функцію належності нечіткій множині, а з її допомогою кожному проекту поставити у відповідність певне значення з проміжку [0;1]. Це значення називається ступенем належності проекту до нечіткої підмножини найбільш надійних інвестиційних проектів. У прикладному аспекті ступінь належності проекту до вказаної нечіткої множини може відображати суб'єктивну міру того, наскільки цей проект відповідає поняттю найбільш надійного (з позиції інвестора).

Припустимо, що відомі величини μ_{kj} — значення функцій належності k-го проекту (k = 1,…, m) до нечіткої підмножини найбільш надійних інвестиційних проектів в умовах j-го стану економічного середовища (j = 1,…, n). У розгорнутому вигляді ситуація прийняття інвестиційного рішення характеризується матрицею (функціоналом оцінювання):

$$M = M^{+} = \left(\mu_{kj}^{+} : k = 1,\ldots,\ m;\ j = 1,\ldots,n\right) = \begin{pmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1n} \\ \mu_{21} & \mu_{22} & \cdots & \mu_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \mu_{m1} & \mu_{m2} & \cdots & \mu_{mn} \end{pmatrix},$$

де елементи $\mu_{kj}^{+} \in [0;1] (k = 1,\ldots,m;\ j = 1,\ldots,n)$,при цьому μ_{kj} – це суб'єктивна міра надійності k-го проекту, тобто μ_{kj} – це міра степеня належності k-го проекту до портфеля найбільш надійних проектів у разі реалізації j-го стану економічного середовища.

Нехай парна гра з нульовою сумою, що визначається матрицею платежів $M = (\mu_{kj})$, не має сідлової точки. Тоді розв'язком цієї гри є оптимальна мішана стратегія першого гравця, що їй відповідає вектор $P^* = (p_1^*;\ldots;\ p)$, компонента p_k^* якого є питомою вагою k-го проекту в структурі портфеля.

1.4 Ігри з природою

Планування структури посівних площ

Нехай аграрне підприємство (перший гравець) може посіяти одну з трьох культур. Його стратегії позначимо через s_1, s_2, s_3. Необхідно визначити, яку з культур сіяти, якщо за інших рівних умов урожаї цих культур залежать, головним чином, від погоди (Θ), а план посіву має забезпечити найбільший прибуток. Уважатимемо, що сільськогосподарське підприємство має надійний спосіб прогнозування погоди. Визначаємо для другого гравця (“погода”) такі стани (стратегії): $Θ_1$ — рік посушливий; $Θ_2$ — рік нормальний; $Θ_3$ — рік дощовий.

Нехай на основі досвіду відомо, що за сухої погоди з 1 га можна зняти h_{k1} центнерів культури s_k за нормальної — h_{k2}, за дощової — h_{k3} (k = 1, 2, 3,). Нехай також відомі ціни: c_k — ціна 1ц культури s_k (k = 1, 2, 3) в умовних грошових одиницях (УГО). Приймемо, що:

$$f_{kj} = c_k h_{kj}, \quad k = 1, 2, 3; \quad j = 1, 2, 3.$$

Якщо знехтувати вартістю насіння і витратами на обробіток ґрунту, отримуємо функціонал оцінювання

$$F^+ = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix},$$

тобто матрицю валових доходів підприємства від реалізації своєї продукції з 1 га за всіх можливих ситуацій. Нехай гра не має сідлової точки і перший гравець (аграрне підприємство) має хоча б одну оптимальну мішану стратеію s_{p^*}.

Якщо V* — ціна гри, то для мішаної стратегії P* виконується нерівність:

$$f_{1j}p_1^* + f_{2j}p_2^* + f_{3j}p_3^* \geq V^*.$$

Очевидно, що ціна гри V* є величиною очікуваного валового доходу з 1 га за j-го стану погоди, якщо підприємство p_1*-ту частку 1 га засіє культурою s_1, p_2*-ту частку 1 га — культурою s_2, а p_3*-ту частку 1 га — культурою s_3.

Отже, засіявши поле культурами s_1, s_2, s_3 у пропорції p_1*, p_2*, p_3*, аграрне підприємство отримає за всіх погодних умов очікуваний валовий дохід, не менший

числа V*. Зауважимо, що очікуваний валовий дохід з 1 га за j-го стану погоди буде принципово відмінним від фактичного, який є реалізацію випадкової величини A саме, за умови реалізації j-го стану погоди, підприємство, реалізувавши мішану стратегію s_p*, одержить з імовірністю p_1* фактичний валовий дохід f_{ij}; з імовірністю p_2* — f_{2j}; з імовірністю p_3* — f_{3j}. Проте відповідно до закону великих чисел фактичний валовий дохід за кілька років з великою ймовірністю дорівнюватиме очікуваному валового доходу V*.

Викладений тут результат легко узагальнити на випадок, коли висіваються типи культур, а стани погоди деталізовано. Крім того, аналогічні моделі можна побудувати для випадку, коли підприємство має можливість змінювати не лише культури, які воно висіває, а й способи (технології) обробки поля.

Розв’яжемо числовий приклад для даних, наведених у табл. 1.3.1.

Отже, функціонал оцінювання (матриця виграшу першого гравця) має вигляд:

$$F = F^{+} = \begin{pmatrix} 40 & 10 & 30 \\ 30 & 50 & 20 \\ 0 & 60 & 80 \end{pmatrix}$$

Таблиця 1.3.1

Стратегія першого гравця (аграрне підприємство)		Стратегія другого гравця (погода)			Ціна за 1ц в УГО
		Суха погода θ_1	Нормальна погода θ_1	Дощова погода θ_1	
Урожайність першої культури, ц/га	s_1	20	5	15	2
Урожайність другої культури, ц/га	s_2	7,5	12,5	5	4
Урожайність третьої культури, ц/га	s_3	0	7,5	10	8

Оскільки $\alpha < \beta$, то гра не має сідлової точки, а тому оптимальна стратегія першого гравця мішана. Для знаходження такої стратегії треба розв’язати задачу лінійного програмування:

$$Z=(t_1+t_2+t_3)=\frac{1}{V}\to \min_{t_1,t_2,t_3}$$

за виконання умов

$$40t_1+30t_2\geq 1 \ ;$$

$$10t_1+50t_2+60t_3\geq 1 \ ;$$

$$30t_1+20t_2+50t_3\geq 1 \ ;$$

$$t_1, t_2, t_3\geq 0 \ ,$$

де

$$t_1=\frac{P_1}{V} \ ; \ t_2=\frac{P_2}{V} \ ; \ t_3=\frac{P_3}{V} \ ; \ t_1+t_2+t_3=\frac{1}{V} \ .$$

Розв’язавши цю задачу, одержимо, що

$$P^*=(0{,}49\,;0{,}40\,;0{,}11) \ ; \ V^*=31{,}5 \ .$$

Тобто, за решти рівних умов, засіявши 49% поля першою культурою, 40% — другою, 11% — третьою культурою, аграрне підприємство отримає в середньому за низку років за різних погодних умов очікуваний максимальний валовий дохід не менший 31,5 ум. од. за рік.

Використання теорії матричних ігор в військово-морській справі

Розподіл ресурсів і обгрунтування систем озброєння є найбільш розповсюдженими ситуаціями в воєнно-морській справі. Ситуації даного типу складаються тоді, коли наявних ресурсів (виділеного наряду сил, зброї, технічних засобів і матеріально-технічного оснащення) не вистачає для виконання кожнної із запланованих дій найбільш ефективним шляхом або коли невідомі кокретні значення некерованих параметрів, а також їх можливий розподіл. В даній проблемі при багаточисельності матеріальних зв’язків і обмежень вирішальне значення мають інтереси сторін, які беруть участь, проінформованість їх про оточуюче середовище та одне про одного. Отже, по своїй суті, ситуації розподілу ресурсів і обгрунтування систем озброєння є конфліктними, формалізація їх природнім шляхом здійснюється на мові теорії ігор.

Зазвичай, сторони, які беруть участь в конфліктній ситуації розподільчого типу, переслідують протилежні цілі, тому їх математичними моделями можуть бути антагоністичні ігри. В тих випадках, коли розподіл ресурсів має скінченне число варіантів (стратегій), для побудови моделей і їх розв'язування використовується теорія матричних ігор.

Побудова теоретико-ігрової моделі двостороннього розподілу ресурсів, її розв'язування і реалізація отриманого розв'язку в процесі розробки практичних рекомендацій здійснюється, як викладено в попередніх параграфах. При цьому сила теоретико-ігрового підходу (як і будь-якого іншого математичного методу) до вибору оптимального варіанта розподілу ресурсів чи оптимальної системи озброєння полягає в спільності та широті застосування, що наглядно ілюструються наступними прикладами практичного (воєнно-морського), хоча, можливо і умовного значення.

Задача про побудову протиповітряної оборони військово-морської бази

Припустимо, що для організації протиповітряної оборони (ПВО) воійськово-морської бази (ВМБ) можна використовувати два типа рухомих систем зенітних ракетних комплексів(ЗРК-1 і ЗРК-2). Система ЗРК-1 призначена для знищення високолітаючих повітряних цілей, а ЗРК-2 – для знищення низьколітаючих цілей. Потрібно визначити оптимальний склад комлексів з врахуванням тактики дій повітряного противника, який може атакувати ВМБ як на малих, так і на великих висотах.

Очевидно, що для заданих умов оптимізація можлива тільки з позиції теоретико-ігрового підходу, тобто шляхом використання принципів оптимальності теорії ігор та максимінного критерія. Для визначення оптимального складу зенітних комплексів потрібно побудувати теоретико-ігрову модель, а саме матричну гру.

Доволі просто встановити, що сторона А(гравець 1) для організації ПВО може використовувати ЗРК-1 (1-ша стратегія) чи ЗРК-2 (2-га стратегія), а сторона В (гравець 2) може атакувати воєнно-морську базу на великих висотах (1-ша стратегія) чи на малих висотах (2-га стратегія). Нехай можливість ураження літака сторони В в

залежності від висоти його польоту тією чи іншою системою ЗРК є такою, що матриця виграшів гравця 1 має вигляд (див. табл. 1.3.2).

Таблиця 1.3.2

Теоретико-ігрова модель розподілу комплексів

	1	*2*	$\min_i a_{ij}$
1	*0,8*	*0,2*	*0,2*
2	*0,4*	*0,6*	*0,4 (максимін)*
$\max_i a_{ij}$	*0,8*	*0,6 (мінімакс)*	

Із таблиці 1.3.2 видно, що максимін не рівний мінімаксу, тому розв'язок гри існує лише в мішаних стратегіях. Знаходимо розв`язок гри:

$$x^* = (1/4, 3/4) \;\; ; \;\; y^* = (1/2, 1/2) \;\; ; \;\; v = 0{,}5 \;\; .$$

Для реалізції отриманого розв'язку використаємо поняття "фізична суміш стратегій". Параметрами цієї стратегії є пропорції, в яких змішується окремі зразки військової техніки. Для даного прикладу під фізичною сумішшю стратегій потрібно розуміти структуру ПВО, яка включає в себе зенітні комплекси ЗРК-1 та ЗРК-2 в відношенні 1:3. Але фізична суміш стратегій являє собою чисту стратегію, тому що вона рекомендується для використання з ймовірністю рівній одиниці. В зв'язку з цим резонно твердження , що фізична суміш стратегій повинна бути представлена як одна із чисельних стратегій в матриці виграшів гравця 1. Доповнимо матрицю гри фізичними сумішами стратегій гравців 1 та 2, а потім знайдемо розв'язок нової гри (див. табл. 1.3.3).

Із таблиці 1.3.3 видн, що 3*3 гра має сідлову точку *(x*,y*)*. Отже, стратегії *x** та *y** являються єдиними стратегіями, які складають розв'язок як нової, так і вихідної гри. Слід зазначити, що отриманий результат не є випадковим співпадінням, а є закономірним для будь-якої матриччної гри, якщо тільки фізична суміш стратегій має реальний зміст.

Таблиця 1.3.3

Теоретико-ігрова модель розподілу комплексів

	1	*2*	*y*=(1/2;1/2)*	$\min_i a_{ij}$
1	*0,8*	*0,2*	*0,5*	*0,2*
2	*0,4*	*0,6*	*0,5*	*0,4*
x=(1/4;3/4)*	*0,5*	*0,5*	*0,5*	*0,5 (минимакс)*
$\max_i a_{ij}$	*0,8*	*0,6*	*0,5 (минимакс)*	

Слід зазначити, що повітряного ворога можна розглядати як деякий некерований (випадковий) параметр, який впливає на ефективність системи ПВО військового-морської бази і ймовірнісний розподіл якого невідомо. В такому сенсі пропорція ЗРК-1 та ЗРК-2 вираховується для найменш сприятливих розподілів *y** значень даного параметра. Розглядаючи вибір висот для літаків, атакуючих військово-морську базу, пропорцію ЗРК-1 та ЗРК-2 чи їх кількість можна приймати за некерований параметр ("природу"), який впливає на ефективність ПВО, і визначити оптимальні висоти вже в сенсі найменш сприятливого розподілу даного параметру.

Задача про транспортні кораблі

Сторона В здійснює перевезення вантажів на одиночних транспортах, а також організовує конвої, які залежно від складу діляться на малі та великі. Для зриву перевезень сторона А використовує підводні човни, які діють по одинці і завісами. Нехай за деякий період часу сторона А знищила число транспортів сторони В, яке наведене в табл. 1.3.4

Таблиця 1.3.4

Число знищених транспортів (матриця виграшів гравця 1)

		Способи дій сторони Б (стратегії гравця II)			
		Одиночні трнспорти 1	Малі конвої 2	Великі конвої 3	$\min_j a_{ij}$
Способи дій сторони А (стратегії гравця 1)	Одиночні пл 1	8	4	2	2
	Завіси 2	3	6	10	3 (максимін)
$\max_i a_{ij}$		8	6 (міні-макс)	10	

Потрібно на підставі наявних статистичних даних визначити відсоткове співвідношення числа підводних човнів, які доцільно використовувати поодинці і в завісах, а також співвідношення числа транспортів, яким доцільно здійснювати перехід самостійно і в складі малих та великих конвоїв.

З табл. 1.3.4 видно, що підводні човни, діючи в завісах, потопили 19 транспортів, а діючи поодинці 14 транспортів. При цьому сторона В приблизно втрачала однакове число транспортів, які йшли самостійно і у складі конвоїв. Виходячи з цього, здавалося б, стороні А вигідно використовувати підводні човни тільки в завісах, а стороні В зберегти структуру перевезень. Однак таке міркування не враховує можливої зміни в діях противника. Очевидно, що використання тільки завіс викличе вагоме збільшення перевезень вантажів на одиночних транспортах, що вимагатиме організації одиночних дій підводних човнів і т. д. Отже, необхідний

аналіз двостороннього розподілу, тобто на основі теоретико-ігрового підходу. Для цього побудуємо 2*3-гру (табл. 1.3.4) і знайдемо оптимальні стратегії гравців.

Шляхом графічного побудови знаходимо, що $y_3^* = 0$. В результаті отримано гру з матрицею виграшем гравця 1

$$\begin{pmatrix} 8 & 4 \\ 3 & 6 \end{pmatrix}$$

і знайдемо $x^* = (3/7, 4/7)$; $y^0 = (2/7, 5/7)$; $v = 36/7$. Оптимальна стратегія y^* для гравця 2 в 2*3-грі виходить з вектора y^0 через додавання нульової компоненти, яка відповідає третій стратегії, тобто $y^* = (2/7, 5/7, 0)$.

Таким чином , на підставі теоретико-ігрового аналізу можна зробити висновок, що стороні А доцільно використовувати підводні човни поодинці і в завісах в відношенні 3:4, а стороні В здійснювати перевезення на одиночних транспортах і малими конвоями в відношені 2:5. В цьому випадку можна вважати, що ефек тивність такого двостороннього розподілу сил еквівалентна математичному сподіванню виграшу в наведеній грі.

Зрозуміло, розглядаючи розподіл підводних човнів (так само як і розподіл транспортів), правомірно приймати протилежну сторону за випадковий параметр, розподіл якого невідомо, і визначати оптимальний спосіб для умов найменш сприятливого розподілу цього параметра.

1.5 Критика методу мішаних стратегій і перехід до нечіткості

У 1994 році Нобелівська премія з економіки була присуджена Джону Нешу «за аналіз рівноваги в теорії некоаліційних ігор». До виходу в світ дисертації Джона Неша дослідження проводились основному в області ігор з нульовою сумою, де сумарний виграш одних учасників дорівнює сумарному програшу інших. Неш досліджував гри з ненульовою сумою, в яких можливі ситуації, вигідні відразу для всіх учасників. Так, наприклад, у разі суперечки керівництва заводу і профспілки про підвищення заробітної плати вона може закінчитися страйком, що не вигідно відразу обом учасникам суперечки, тоді як від певної угоди всі учасники конфлікту можуть в цілому залишитися у виграші. Неш виділив спеціальний клас ситуацій в грі (тобто фіксованих стратегій всіх учасників гри) - такі ситуації, від яких не вигідно відхилятися поодинці нікому з гравців. Пізніше такі ситуації були названі ситуаціями рівноваги по Нешу.

В чистих стратегіях в грі може не існувати ситуації рівноваги. Орієнтація на оптимальні мішані стратегії дозволяє першому гравцеві максимізувати найменший очікуваний виграш, а другому мінімізувати найбільший очікуваний програш. Однак такий вибір може бути виправданий лише для багаторазово повторюваних ігор. В окремій грі кожен з гравців навіть при реалізації «оптимальної» мішаної стратегії може дуже сильно програти. В окремих іграх мішані стратегії не тільки не обіцяють максимального виграшу, але навіть не захищають від максимально можливого програшу, тобто кожен гравець може прийняти найгірше з усіх можливих рішень!

Вибір максимінної і мінімаксної стратегій добре підходить для обережних гравців. Він дозволяє не програти занадто багато, не виграти занадто мало. Але деякі гравці можуть бути більш схильні до ризику, і орієнтуватися на ті стратегії, які можуть забезпечити їм максимально можливий виграш, деякі намагаються максимізувати середній очікуваний виграш, тобто вибір мінімаксу або максиміну не є єдиноможливим хорошим в деякому сенсі рішенням.

Нарешті, орієнтація на мішані стратегії передбачає відсутність будь-якої інформації про психологію суперника, припускаючи лише, що він діє раціонально.

Однак, як зауважив Герберт Саймон у своїй теорії обмеженої раціональності, більшість людей раціональні тільки частково й емоційні або ірраціональні в інших ситуаціях. Вибір раціонального рішення нерідко несе в собі складні обчислення, якими гравці можуть не володіти або ж вони можуть не мати можливості їх здійснити (через трудомісткість обчислень, відсутності всієї необхідної інформації або яких-небудь ще причин). Можливо також, що витрати на прийняття оптимального рішення можуть бути неприпустимо високі і не окуплять можливих переваг, тому раціонально в такій ситуації буде прийняти будь-яке інше «досить хороше» рішення.

На практиці ми можемо мати про противника будь-яку додаткову інформацію і на її основі робити які-небудь припущення щодо його поведінки. При цьому наші припущення практично завжди носять нечіткий характер. Приміром, наші припущення можуть мати такий вигляд: «якщо противник самовпевнений, то, скоріше за все, він вибере стратегію максимізації максимального доходу».

Оперувати з подібними висловлюваннями і приймати на їх основі кількісне раціональне рішення допомагає апарат створеної Лотфі Заде теорії нечітких множин. На відміну від класичної теорії множин, де кожен елемент може або належати, або не належати якійсь множині, в теорії нечітких множин передбачається, що будь-який елемент x належить безлічі A з деяким ступенем приналежності $\mu_A(x) \in [0,1]$. Пара $(A, \mu_A(x))$ утворює нечітку множину. Незважаючи на удавану подібність з ймовірнісної мірою, функція приналежності за своєю природою має зовсім інший характер. Так, якщо б ми з імовірністю 0,7 припускали, що наш супротивник схильний до ризику, то це означало б лише, що з 100 зустрічашихся нам схожих в якомусь сенсі супротивників приблизно 70 були схильні до ризику. Такі дані є хорошими в умовах повної невизначеності щодо нашого противника і при наявності необхідної статистики. Ступінь приналежності рівна 0,7 до множини супротивників схильних до ризику можна інтерпретувати приблизно, так, якщо супротивник скоріше схильний до ризику, ніж не схильний, але впевненість у цьому не повна. Така впевненість хоч і суб'єктивна, але зате не передбачає наявності будь-яких

статистичних даних, враховує конкретного супротивника і, так чи інакше, формалізує всю сукупність наших знань про нього і нашого досвіду. Якщо ми граємо з дитиною або домогосподаркою, або ж ми граємо з яких-небудь вченим, досвідченим гравцем, фахівцем в цій галузі, то такі знання нерозумно не враховувати, хоча вони, звичайно ж, не гарантують правильність наших припущень. Часто ми, так чи інакше, знаємо нашого противника і можемо, хоча б на інтуїтивному рівні, припускати, що він буде керуватися при прийнятті рішення.

У загальному випадку наші припущення можуть носити досить складний характер. Наприклад, ми можемо припускати, що «якщо противник досвідчений і поводиться самовпевнено, то він буде керуватися принципом оптимальних мішаних стратегій, якщо випала стратегія буде не занадто поганою». Якщо супротивник обережний, то він буде керуватися максимінна стратегією. Якщо супротивник схильний до ризику і не досвідчений, то він вибере стратегію, відповідну максимально можливого доходу. Таким чином, ми можемо скласти цілу базу нечітких правил. При цьому нам необхідно евристично або аналітично задати ступені приналежності супротивника до кожної з нечітких множин. Також необхідно навчитися здійснювати логічні операції над нечіткими множинами. Так, у першому прикладі нам необхідно обробити цілих три нечітких множини: A = {досвідчений противник}, B = {самовпевнений противник} і C = {не дуже погана стратегія} і на основі їх обробки здійснити процедуру нечіткого виводу. Методи обробки подібних висловлювань будуть розглянуті в підрозділі 2.4.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Розв’язування матричної антагоністичної гри, приведеної до задачі лінійного програмування

На вході маємо задачу.

Завдання другого гравця мінімізація програшу V	Завдання першого гравця максимізація виграшу V
Цільова функція	
$F' = x_1 + x_2 + x_n = \frac{1}{V} \to max$	$F = y_1 + y_2 + ... + y_m = \frac{1}{V} \to min$
Функціональні обмеження	
$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leqslant 1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leqslant 1 \\ ..... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leqslant 1 \end{cases}$	$\begin{cases} a_{11}y_1 + a_{21}y_2 + ... + a_{m1}y_m \geqslant 1 \\ a_{12}y_1 + a_{22}y_2 + ... + a_{m2}y_m \geqslant 1 \\ ..... \\ a_{1n}y_1 + a_{2n}y_2 + ... + a_{mn}y_m \geqslant 1 \end{cases}$
Загальні (прямі) обмеження	
$x_1 \geq 0, x_2 \geq 0, .., x_n \geq 0$	$y_1 \geq 0, y_2 \geq 0, .., y_m \geq 0$

Завдання обох гравців утворюють пару симетричних взаємодвоїстих задач лінійного програмування. Немає необхідності розв'язувати обидві ці задачі, тому що знайшовши розв’язок однієї з них, можна знайти і розв’язок іншої.

Примітка.

З процедури приведення скінченної матричної антагоністичної гри до задачі лінійного програмування (ЗЛП) можна зробити висновок з приводу існування розв’язку антагоністичної гри m × n, не спираючись на теорему фон Неймана.

Нехай задача про знаходження оптимальної стратегії $P^* = (p_1^*, p_2^*, ..., p_m^*)$ гравця А зведена до задачі лінійного програмування з умовами нерівностями (2.1.4) і мінімізується функцією (2.1.5). Чи завжди існує її розв’язок? Відомо, що розв’язок задачі лінійного програмування може і не існувати; воно відсутнє, якщо:

1. функціональні та загальні умови - рівності або нерівності - взагалі не

мають допустимих невід'ємних розв’язків;

2. допустимі розв’язки існують, але серед них немає оптимального, так як мінімізується функція не обмежена знизу.

Подивимося, що буде в нашому випадку. Допустимі розв’язки ЗЛП в нашому випадку завжди існують. Дійсно, за допомогою афінних перетворень зробимо елементи платіжної матриці А строго позитивними, наприклад, додавши досить велике позитивне число М до кожного елементу платіжної матриці, і позначимо найменший елемент матриці А через μ**:**

$$\mu = \min_i \min_j a_{ij}$$

Перший опорний план можна записати як $y_1 = \frac{1}{\mu}$, $y_2 = y_3 = ... = y_m = 0$. Неважко побачити, що ця система значень змінних $y_1, y_2, y_3, ..., y_m$ є допустимим розв’язком ЗЛП - всі вони невід’ємні, і їх сукупність задовольняє умовам (2.1.4).

Тепер переконаємося, що лінійна функція (2.1.5) не може бути необмеженою знизу. Дійсно, всі $y_1, y_2, y_3, ..., y_m$ невід’ємні, а коефіцієнти при них в у виразі (2.1.5) позитивні (рівні одиницям), значить, функція F у формулі (2.1.5) теж невід’ємна, значить, вона обмежена знизу (нулем) і розв’язок задачі лінійного програмування (а отже, і ігри $m \times n$) існує.

Приклад. Розв’язати гру з матрицею:

$$\begin{pmatrix} 0 & 0,3 & 0,5 \\ 0,6 & 0,1 & -0,1 \\ 0,4 & 0,2 & 0,1 \end{pmatrix}$$

1. Відповідно до алгоритму визначимо, чи існують в ній доміновані стратегії, щоб виключити їх. Домінованих стратегій немає.

2. Оскільки матриця містить негативні числа, то потрібно зробити так, щоб усі її елементи були невід’ємні, додавши до всіх її елементів число, рівне модулю найменшого числа матриці. Мінімальний елемент матриці дорівнює -0,1, його модуль дорівнює 0,1. Додамо до всіх елементів платіжної матриці число, рівне 0,1, в результаті отримаємо:

$$\begin{pmatrix} 0,1 & 0,4 & 0,6 \\ 0,7 & 0,2 & 0 \\ 0,5 & 0,3 & 0,2 \end{pmatrix}$$

Помножимо всі елементи отриманої матриці на 10, щоб зручніше проводити подальші обчислення.

$$\begin{pmatrix} 1 & 4 & 6 \\ 7 & 2 & 0 \\ 5 & 3 & 2 \end{pmatrix}$$

Проведені афінні перетворення на оптимальних стратегіях не позначаться, а ціну гри ми відновимо, зробивши зворотні перетворення (розділю отриману суму на 10 і віднявши 0,1).

Припишемо рядкам ймовірності p_1, p_2, p_3

$$\begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}\begin{pmatrix} 1 & 4 & 6 \\ 7 & 2 & 0 \\ 5 & 3 & 2 \end{pmatrix}$$

Тоді середнє значення (математичне очікування) виграшу гравця А при застосуванні гравцем В своєї першої стратегії дорівнює $1p_1 + 7p_2 + 5p_3$ (перший стовпець поелементно множиться на ймовірності p_1, p_2, p_3 та отримані добутки підсумовуються). Цей виграш не може бути менший гарантованої ціни гри V:

$$1p_1 + 7p_2 + 5p_3 \geq V.$$

Аналогічно для інших стратегій гравця В.

$$\begin{cases} p_1 + 7p_2 + 5p_3 \geq V \\ 4p_1 + 2p_2 + 3p_3 \geq V \\ 6p_1 + 2p_3 \geq V \\ p_i \geq 0, i = 1,2,3 \end{cases}$$

Розділимо обидві частини нерівності на V і введемо позначення:

$$y_i = p_i / V \ (i = 1, 2, 3)$$

$$\begin{cases} y_1 + 7y_2 + 5y_3 \geq 1 \\ 4y_1 + 2y_2 + 3p_3 \geq 1 \\ 6y_1 + 2y_3 \geq 1 \\ y_i \geq 0, i = 1,2,3 \end{cases}$$

$$F = y_1 + y_2 + y_3 = (p_1 + p_2 + p_3)/V = \frac{1}{V}$$

Гравець А прагне підвищити ціну гри (V → max). Тому F → min.

Отримано завдання лінійного програмування:

$$F = y_1 + y_2 + y_3 \rightarrow \min$$

$$\begin{cases} y_1 + 7y_2 + 5y_3 \geqslant 1 \\ 4y_1 + 2y_2 + 3p_3 \geqslant 1 \\ 6y_1 + 2y_3 \geqslant 1 \\ y_i \geqslant 0, i = 1,2,3 \end{cases}$$

Аналогічно припишемо стовпцям ймовірності q_1, q_2, q_3.

$$\begin{array}{c} q_1 \quad q_2 \quad q_3 \\ \begin{pmatrix} 1 & 4 & 6 \\ 7 & 2 & 0 \\ 5 & 3 & 2 \end{pmatrix} \end{array}$$

Тоді середній програш гравця В при застосуванні гравцем А його першої стратегії дорівнює $1q_1 + 4q_2 + 6q_3$ (1-й рядок поелементно множиться на ймовірності q_1, q_2, q_3 та отримані добутки підсумовуються). Цей програш не може бути більше ціни гри V: $1q_1 + 4q_2 + 6q_3 \leq V$. Аналогічно для інших стратегій гравця А.

$$\begin{cases} q_1 + 4q_2 + 6q_3 \leqslant V \\ 7q_1 + 2q_2 + 0q_3 \leqslant V \\ 5q_1 + 3q_2 + 2q_3 \leqslant V \\ q_i \geqslant 0, i = 1,2,3 \end{cases}$$

Розділимо обидві частини нерівностей на V і введемо позначення

$$x_i = q_i / V \ (i = 1, 2, 3)$$

$$\begin{cases} x_1 + 4x_2 + 6x_3 \leqslant 1 \\ 7x_1 + 2x_2 + 0x_3 \leqslant 1 \\ 5x_1 + 3x_2 + 2x_3 \leqslant 1 \\ x_i \geqslant 0, i = 1,2,3 \end{cases}$$

$$F' = x_1 + x_2 + x_3 = (q_1 + q_2 + q_3)/V = \frac{1}{V}$$

Гравець В прагне знизити ціну гри (V → min), тому $F^{/} \rightarrow max$.

Отримано задачу лінійного програмування:

$$F' = x_1 + x_2 + x_3 \rightarrow max$$

$$\begin{cases} x_1 + 4x_2 + 6x_3 \leqslant 1 \\ 7x_1 + 2x_2 + 0x_3 \leqslant 1 \\ 5x_1 + 3x_2 + 2x_3 \leqslant 1 \\ x_i \geqslant 0, i = 1,2,3 \end{cases}$$

Отримані задачі є взаємнодвоїстими задачами лінійного програмування. Будь-яку з них можна розв'язати симплекс-методом . Остаточний результат має такий вигляд:

$y_1 = 3/28$; $p_1 = 3/8$; $x_1 = 1/7$; $q_1 = 1/2$;

$y_1 = 0$; $p_1 = 0$; $x_2 = 0$; $q_2 = 0$;

$y_1 = 5/28$; $p_1 = 5/8$; $x_3 = 1/7$; $q_3 = 1/2$;

$F^{/} = 2/7$; $V = 3\frac{1}{2}$; $F = 2/7$; $V = 3\frac{1}{2}$;

Отже, оптимальні стратегії:

$P^* = (3/8; 0; 5/8)$, $Q^* = (1/2; 0; 1/2)$,

ціна гри для модифікованої задачі V = 3,5,

а для вихідної завдання $V^{/} = 3,5 / 10-0,1 = 0,25$.

2.2 Використання методу Брауна-Робінсона в розв'язанні матричних ігор

Припустимо, що за перші k розігрування гравець 1 використовував і-ю чисту стратегію ξ_i^k разів (i=1,…,m), а гравець 2 j-ю чисту стратегію η_j^k разів (j=1,…,n). Тоді їх мішаними стратегіями будуть вектори $x^k = (\xi_1^k/k,...,\xi_m^k/k), y^k = (\eta_1^k/k,...,\eta_n^k/k)$.

Гравець 1 стежить за діями гравця 2 і з кожним своїм ходом бажає отримати якомога більший виграш. Тому у відповідь на застосування гравцем 2 своєї мішаної стратегії y_k,він буде використовувати чисту стратегію i_k +1,яка забезпечить йому найкращий результат при розігруванні (k +1)-ої партії. Гравець 2 надходить аналогічно. В гіршому випадку кожен з них може отримати:

$$\bar{v}^k = \max_i \sum_j^n a_{ij}\eta_j^k = \sum_j^n a_{i_{k+1}}\eta_j^k$$

$$\underline{v}^k = \min_j \sum_i^m a_{ij}\xi_i^k = \sum_i^m a_{ij_{k+1}}\xi_i^k$$

де $\bar{v}^k$ - найбільше значення сумарного програшу гравця 2 і $\underline{v}^k$ - найменше значення сумарного виграшу гравця 1 за k партій.

Розглянемо відношення, які визначають середні значення програшу гравця 2 і виграшу гравця 1:

$$\bar{v}^k/k = \max_i \sum_j^n a_{ij}\eta_j^k/k = \sum_j^n a_{i_{k+1}j}\eta_j^k/k$$

$$\underline{v}^k/k = \min_j \sum_i^m a_{ij}\xi_i^k/k = \sum_i^m a_{ij_{k+1}}\xi_i^k/k$$

Нехай ν - ціна матричної гри $Г_A$. Її значення буде більше виграшу гравця 1, але менше програшу гравця 2, тобто

$$\max_k \underline{v}^k/k \le \nu \le \min_k \bar{v}^k/k. \qquad (3.2.1)$$

Таким чином, отриманий ітеративний процес, що дозволяє знаходити наближений розв'язок матричної гри, при цьому ступінь близькості наближення до істинного значення гри визначається довжиною інтервалу

$$\left[\max_k \underline{v}^k/k, \min_k \bar{v}^k/k\right].$$

Збіжність алгоритму гарантується наступною теоремою.

Теорема. $\lim\limits_{k\to\infty}\left(\min\limits_{k}\bar{v}^{k}/k\right)=\lim\limits_{k\to\infty}\left(\max\limits_{k}\underline{v}^{k}/k\right)=v$.

Схема доказу.

Лемма. Для будь-якої матриці A і $\forall\varepsilon>0$ існує таке k_0, що

$$\min_{k_0}\bar{v}^{k}/k-\max_{k_0}\underline{v}^{k}/k<\varepsilon .$$

При граничному переході в нерівності (3.2.1) при $k\to\infty$ маємо:

$$\lim_{k\to\infty}\left(\max_{k}\underline{v}^{k}/k\right)\le v\le\lim_{k\to\infty}\left(\min_{k}\bar{v}^{k}/k\right) . \qquad (3.2.2)$$

Звідси отримуємо оцінку різниці границь:

$$0\le\lim_{k\to\infty}\left(\min_{k}\bar{v}^{k}/k\right)-\lim_{k\to\infty}\left(\max_{k}\underline{v}^{k}/k\right) .$$

З леми випливає, що

$$\lim_{k\to\infty}\left(\min_{k}\bar{v}^{k}/k-\max_{k}\underline{v}^{k}/k\right)<\varepsilon .$$

На основі нерівності (1) маємо:: $\begin{matrix}\lim\limits_{k\to\infty}\inf\left(\min\limits_{k}\bar{v}^{k}/k\right)\ge v\\ \lim\limits_{k\to\infty}\sup\left(\max\limits_{k}\underline{v}^{k}/k\right)\le v\end{matrix}$.

Отже, в силу обмеженості границь

$$\lim_{k\to\infty}\left(\min_{k}\bar{v}^{k}/k\right)-\lim_{k\to\infty}\left(\max_{k}\underline{v}^{k}/k\right)<\varepsilon .$$

Отримуємо оцінку для різниці меж:

$$0\le\lim_{k\to\infty}\left(\min_{k}\bar{v}^{k}/k\right)-\lim_{k\to\infty}\left(\max_{k}\underline{v}^{k}/k\right)<\varepsilon \quad \text{для } \forall\varepsilon>0.$$

Можемо зробити висновок, що $\lim\limits_{k\to\infty}\left(\min\limits_{k}\bar{v}^{k}/k\right)=\lim\limits_{k\to\infty}\left(\max\limits_{k}\underline{v}^{k}/k\right)$. Залишилося показати рівність границь v. Це випливає з нерівності (3.2.2).

Отже, $\lim\limits_{k\to\infty}\left(\min\limits_{k}\bar{v}^{k}/k\right)=\lim\limits_{k\to\infty}\left(\max\limits_{k}\underline{v}^{k}/k\right)=v$.

Приклад. *Знайти наближений розв'язок гри з матрицею*

$$A=\begin{bmatrix}0 & 3 & 1\\ 4 & 1 & 2\\ 2 & 0 & 3\end{bmatrix}.$$

Нехай гру почне гравець 2. Він довільно обирає одну зі своїх чистих стратегій. Припустимо, що він вибрав свою 1-у стратегію, а гравець 1 відповідає своїй 2-й стратегією. Занесемо дані в таблицю.

Номер партії	Стратегії другого гравця	Виграш гравця 1 при його стратегіях			Стратегії першого гравця	Програш гравця 2 при його стратегіях			υ	ω	ν
		1	2	3		1	2	3			
1	1	0	4	2	2	4	1	2	4	1	5/2

У стовпці υ знаходиться найбільший середній виграш 4 гравця 1, отриманий ним у першій партії; в стовпці ω варто найменший середній програш 1, отриманий гравцем 2 в першій партії; в стовпці ν знаходиться середнє арифметичне ν=(υ+ω)/2=5/2, тобто наближене значення ціни гри, отримане в результаті програвання однієї партії.

Так як гравець 1 вибрав 2-у стратегію , то гравець 2 може програти:

4, якщо застосує свою 1-у стратегію;

1, якщо застосує свою 2-у стратегію;

2, якщо застосує свою 3-ю стратегію.

Оскільки він бажає програти якомога менше, то у відповідь застосує свою 2-у стратегію.

Тоді перший гравець отримає виграш рівний 3, 1, 0 відповідно при своїх 1-й, 2-й, 3-й стратегіях, а його сумарний виграш за дві партії складе:

0 +3 = 3 при його 1 - й стратегії;

4 +1 = 5 при його 2-й стратегії;

2 +0 = 2 при його 3-й стратегії.

З усіх сумарних виграшів найбільшим є 5, який виходить при 2-й стратегії гравця 1. Значить, у цій партії він повинен вибрати саме цю стратегію.

При 1-й стратегії гравця 1 гравець 2 програє 4, 1, 2 відповідно 1-й, 2-й, 3-й його стратегіям, а сумарний програш за обидві партії складе:

4 +4 = 8 при його 1-й стратегії;

1 +1 = 2 при його 2-й стратегії;

2 +2 = 4 при його 3-й стратегії.

Всі отримані дані занесемо в таблицю. У стовпець υ ставиться найбільший сумарний виграш гравця 1 за дві партії, поділений на кількість партій, тобто 5/2; в стовпець ω ставиться найменший сумарний програш гравця 2, поділений на кількість партій, тобто 2/2; в стовпець ν ставиться середнє арифметичне цих значень, тобто 7/2.

Номер партії	Стратегії другого гравця	Виграш гравця 1 при його стратегіях			Стратегії першого гравця	Програш гравця 2 при його стратегіях			υ	ω	ν
		1	2	3		1	2	3			
1 2	1 2	0 3	4 5	2 2	2 2	4 8	1 2	2 4	4 5/2	1 2/2	5/2 7/2

У третій партії гравець 2 вибирає свою 2-у стратегію, так як з усіх сумарних програшів найменшим є 2.

Таким чином, продовжуючи цей процес далі, складемо таблицю розігрування гри за 20 ітерацій (партій).

Номер партії	Стратегії другого гравця	Виграш гравця 1 при його стратегіях			Стратегії першого гравця	Програш гравця 2 при його стратегіях			υ	ω	ν
		1	2	3		1	2	3			
1	1	0	4	2	2	4	1	2	4	1	5/2
2	2	3	5	2	2	8	2	4	5/2	2/2	7/2
3	2	6	6	2	1	8	5	5	6/3	5/3	11/6
4	2	9	7	2	1	8	8	6	9/4	6/4	15/8
5	3	10	9	5	1	8	11	7	10/5	7/5	17/10
6	3	11	11	8	1	8	14	8	11/6	8/6	19/12
7	1	11	15	10	2	12	15	10	15/7	10/7	25/14
8	3	12	17	13	2	16	16	12	17/8	12/8	27/16
9	3	13	19	16	2	20	17	14	19/9	14/9	33/18
10	3	14	21	19	2	24	18	16	21/10	16/10	37/20
11	3	15	23	22	2	28	19	18	23/11	18/11	41/22
12	3	16	25	25	2	32	20	20	25/12	20/12	45/24
13	2	19	26	25	2	36	21	22	26/13	21/13	47/26
14	2	22	27	25	2	40	22	24	27/14	22/14	49/28
15	2	25	27	25	2	44	23	26	27/15	23/15	50/30
16	2	28	29	25	2	48	24	28	29/16	24/16	53/32
17	2	31	30	25	1	48	27	29	31/17	27/17	58/34
18	2	34	31	25	1	48	30	30	34/18	30/18	64/36
19	2	37	32	25	1	48	33	31	37/19	31/19	68/38
20	3	38	34	28	1	48	36	32	38/20	32/20	70/40

З таблиці видно, що в 20-ти програних партіях стратегії 1, 2, 3 для другого гравця зустрічаються відповідно 2, 10, 8 разів, отже, їх відносні частоти рівні 2/20, 10/20, 8/20. Стратегії 1, 2, 3 для гравця 1 зустрічаються відповідно 8, 12, 0 раз, отже, їх відносні частоти рівні 8/20, 12/20, 0, а наближене значення ціни гри одно 70/40.

Таким чином, отримали наближений розв'язок ігри: x_{20}= (1/10, 1/2,2/5),y_{20}= (2/5, 3/5, 0), ν = 1,57.

При цьому точний розв'язок гри симплекс-методом

x^*(0,07; 0,4; 0,53), y^*(0.47; 0.33; 0.2),ν = 1,73.

Такий ітеративний процес веде гравців до мети повільно. Часто для отримання оптимальних стратегій, які дають гравцям виграш, доводиться пробляти сотні ітерацій. При цьому швидкість збіжності помітно погіршується зі зростанням розмірності матриці і зростанням числа стратегій гравців. Це також є наслідком не монотонності послідовностей $\bar{v}^k$ і $\underline{v}^k$. Тому, практична цінність цього методу має місце, коли обчислення проводяться на досить швидкодіючих обчислювальних машинах. Але поряд з таким недоліком можна виділити і переваги методу ітерацій:

1. Цей метод дає можливість знайти орієнтовне значення ціни гри і наближено обчислити оптимальні стратегії гравців.
2. Складність і обсяг обчислень порівняно слабко зростають у міру збільшення числа стратегій гравців (m і n).

2.3 Використання графічного методу в розв'язанні матричних ігор

Ігри розмірності 2*n чи m*2 завжди мають розв'язок, що містить не більше двох активних стратегій для кожного з гравців. Якщо знайти ці активні стратегії, то гра 2*n або m*2 зводиться до гри 2*2. Тому ігри 2*n і m*2 розв'язують зазвичай графоаналітичним методом.

Розглянемо розв'язування матричної гри на прикладі.

Приклад.

$$\begin{pmatrix} 1 & 4 & 7 \\ 6 & 3 & 2 \end{pmatrix}$$

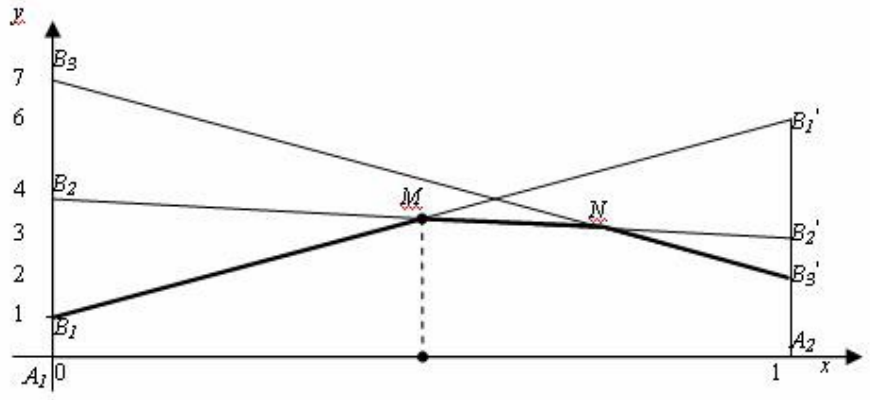
Розв'язування.

				α_i
	1	4	7	1
	6	3	2	2
β_i	6	4	4	2 4

$\alpha = 2$, $\beta = 4$ $\alpha \neq \beta$, Тому гра не має сідлової точки, і розв'язок має бути в мішаних стратегіях.

1. Будуємо графічне зображення гри.

Мал. 1



Якщо гравець B застосує стратегію $B_{1,}$ то виграш гравця A при застосуванні стратегії A_1 дорівнює $a_{11} = 1$, а при використанні A_2 виграш дорівнює $a_{21} = 6$, тому відкладаємо відрізки $A_1B_1 = 1$, $A_2B_1' = 6$ на перпендикулярах в A_1 і A_2 і з'єднуємо їх відрізком. Аналогічно для стратегій B_2 і B_3 будуємо відрізки B_2B_2' і $B_3\,B_3'$.

2. Виділяємо нижню межу виграшу $B_1\,M\,N\,B_3'$ і знаходимо найбільшу ординату цієї нижньої межі, ординату точки *M,* яка дорівнює ціні ігри γ.
3. Визначаємо пару стратегій, що перетинаються в точці оптимуму *M.*

   У цій точці перетинаються відрізки B_2B_2' і B_1B_1', відповідні стратегіям B_1 і B_2 гравця *B*. Отже, стратегію B_3 йому застосовувати невигідно. Виключаємо з матриці третій стовпець і вирішуємо гру 2 х 2 аналітично:

$$\begin{cases} p_1+6\,p_2=\gamma; \\ 4\,p_1+3\mathrm{p}_2=\gamma; \\ p_1+p_2=1. \end{cases}$$

$$p_1+6\,p_2=4\,p_1+3\,p_2;$$
$$3\,p_2=3\,p_1;$$
$$p_1=p_2=\frac{1}{2}$$

$$y=\frac{1}{2}+\frac{6}{2}=\frac{7}{2}$$

$$\begin{cases} q_1+4\,q_2=\dfrac{7}{2}; \\ q_1+q_2=1 \end{cases}$$

$$3\,q_2=\frac{5}{2}\ ,\quad q_2=\frac{5}{6}\ ,\quad q_1=\frac{1}{6}$$

Відповідь: γ = 7/2; P_A = (1/2, 1/2); Q_B = (1/6; 5/6, 0).

2.4 Процедура обробки нечітких правил

Визначимо класичним чином основні логічні операції над нечіткими множинами.

- Об'єднання (відповідає логічній операції «або»):

  $\mu_{A\cup B}= \max [\mu_A(x), \mu_B(x)]$

- Перетин (відповідає логічної операції «і»):

  $\mu_A(x)= \min [\mu_A(x), \mu_B(x)]$

- Доповнення (відповідає логічної операції «не»):

  $\mu_{\bar{A}}(x)= 1 - \mu_{\bar{A}}(x)$

Тоді обробку нечітких правил виду: $\{П_i$: якщо (x є A_i) та (y є B_i), то $z_i\}$, де A_i, B_i - деякі нечіткі множини, а z_i - конкретне рішення можна здійснити наступним чином.

Спочатку оцінимо ступінь нашої впевненості в результаті. Для конкретного супротивника визначаємо ступінь істинності для передумов кожного правила $\mu_{Ai}(x_0)$, $\mu_{Bi}(y_0)$. Для прикладу, впевненість в тому, що супротивник обере стратегію z_i складе $a_i = \min [\mu_{Ai}(x_0), \mu_{Bi}(y_0)]$, так як використовувалася операція диз'юнкції, тобто логічного перетину двох висловлювань.

Потім визначаємо впевненість a_i для кожної з можливих стратегій. Якщо ми вважаємо, що противник вибере стратегію z_i, то оптимальним рішенням з цієї стратегії для нього буде якесь рішення j^*. Оптимальною відповіддю буде вибір рішення $i^*: max_i a_{ij}$.

Тепер можна скористатися *модифікованим варіантом мішаних стратегій*, для вибору оптимальної стратегії, заснованої на обробці бази нечітких правил.

Визначимо ймовірності вибору кожного з допустимих рішень як $p_i=\frac{a_i}{\sum_i a_i}$. Тепер ми можемо вважати, що з імовірністю p_i противник буде керуватися стратегією z_i, згідно з якою він вибере рішення j^*, на яке ми з цією ж вірогідністю відповімо рішенням $i^*: max_i a_{ij}$.

Приклад.

Продемонструємо запропоновану ідею на конкретному прикладі. Припустимо, що як нам, так і нашому супернику відома наступна матриця ігри:

$$M = \begin{bmatrix} 2 & -3 & 4 & -5 \\ -3 & 4 & -5 & 6 \\ 4 & -5 & 6 & -7 \\ -5 & 6 & -7 & 8 \end{bmatrix}$$

З матриці видно, що як ми, так і наш противник повинні вибрати одне з 4х рішень. В кожному рядку матриці є як позитивні, так і негативні виграші, відповідно, дана гра може бути як виграшною, так і програшною для кожного з гравців. Якщо ми спробуємо керуватися максимінною стратегією, то виберемо рішення 1 або рішення 2 (тому що в цьому випадку ми програємо не більше 5 у.о., а при виборі рішення 3 або 4 був ризик програти 7 у.о.). Керуючись мінімаксною стратегією, наш противник вибере рішення 1 (тому що в цьому разі він програє не більше 4 у.о., тоді як при виборі іншого рішення для нього був ризик програти більше). $5 \neq 4$, а значить ситуації рівноваги по Нешу в даній грі немає. Дійсно, при такому виборі можлива реалізація однієї з двох ситуацій: або ми виграємо 2 у.о., або програємо 3. Однак, якщо б ми точно знали, що противник буде керуватися мінімаксною стратегією, то обрали б рішення 3, що гарантувала б нам виграш 4 у.о., тобто зазначені ситуації не є ситуаціями рівноваги, тому відхилятися від них вигідно. Отже, в чистих стратегіях зазначена гра ситуацій рівноваги не має.

Припустимо для прикладу, що ми сформулювали таку базу з 3-х нечітких правил:

$П_1$: {Якщо противник недосвідчений або схильний до ризику, то він вибере рішення за принципом максимізації максимуму можливого доходу}.

$П_2$: {Якщо противник обережний, то він вибере рішення за мінімаксним принципом}.

$П_3$: {Якщо противник досвідчений, то він вибере рішення за мінімаксним принципом, якщо середній виграш при такому рішенні буде не занадто низьким і рішення, максимізуюче середній виграш в іншому випадку}.

Зауважимо, що дана база, звичайно ж, не претендує на повноту. Однак, що є,

то є. Якщо ми вважаємо за необхідне, то можемо доповнити правила.

Задамо відповідні нечіткі множини:

A = {досвідчений супротивник}. Ми можемо вважати, наприклад, що, якщо противник грає в цю гру не більш ніж в 3-ій раз, то він точно недосвідчений, якщо більш ніж в 20-ий - точно досвідчений. Проміжні результати ми можемо, наприклад, апроксимувати за допомогою прямої, задавши функцію приналежності наступним чином:

$$\mu_A(x)=\begin{cases} 0, x<3 \\ \dfrac{x-3}{21-3}, 3<x\leqslant 20 \\ 1, x>20 \end{cases}$$

B = {обережний противник}. Формалізувати конкретно вибір функції приналежності тут досить складно. Але можна виходити при оцінці з наших інтуїтивних (експертних) міркувань. Наприклад, якщо ми точно впевнені, що противник обережний, то задати $\mu_B(y_0) = 1$, якщо до обережних ми його зарахувати ніяк не можемо, то задати $\mu_B(y_0) = 0$, якщо складно сказати, тобто даних, на основі яких ми можемо робити хоч якісь припущення у нас абсолютно недостатньо, то задати $\mu_B(y_0) = 0{,}5$ і так далі.

C = {противник схильний до ризику}. Зауважимо, що противники схильні до ризику необережні і навпаки, тобто можна вважати, що C = $\overline{B}$ (доповнення до множини B).

D = {середній виграш занадто низький}. Наприклад, ми можемо вважати, що середній виграш при вирішенні точно занадто низький, якщо він негативний, і точно не занадто низький, якщо складає більше 2/3 максимально можливого виграшу. Для простоти знову апроксимуємо проміжні значення за допомогою прямій, і поставимо відповідної функції приналежності наступним чином:

$$\mu_D(z)=\begin{cases}1, z\leqslant 0\\ \dfrac{\frac{2}{3}z_{max}-z}{\frac{2}{3}z_{max}-0}, 0<z<\dfrac{2}{3}z_{max}\\ 0, z\geqslant\dfrac{2}{3}z_{max}\end{cases}$$

Розрахуємо значення ступеня приналежності нашого противника до кожної з зазначених нечітких множин. Нехай ми знаємо, що противник грає в 9-ий раз. Тоді

$$\mu_A(x_0)=\frac{9-3}{21-3}=\frac{1}{3} \quad .$$

Нехай ми припускаємо, що противник скоріше обережний, ніжні, але точно не упевнені. Тоді припустимо, що $\mu_B(y_0) = 0{,}7$. Нарешті, максимально можливий виграш противника в нашій грі дорівнює $z_{max} = 7$. Якщо він обере мінімаксну стратегію, то, відповідно до цієї стратегії, як було відмічено раніше, оптимальним для нього буде рішення 1. В цьому випадку його середній виграш складе

$$z=\frac{2+(-3)+4+(-5)}{4}=\frac{1}{2}$$

Отже,

$$\mu_D(z)=\frac{\frac{2}{3}\cdot 7-\frac{1}{2}}{\frac{2}{3}\cdot 7}=\frac{25}{28}$$

Тепер розрахуємо ступінь впевненості в кожному з рішень противника.

Для максимізації максимуму доходу будемо мати: $\alpha_1=max[1-\frac{1}{3},1-0{,}7]=\frac{2}{3}\approx 0{,}67$. Ми взяли функцію максимуму, тому що «Або», $1-\frac{1}{3}$ тому що «недосвідчений» - доповнення до досвідченого, 1 - 0.7, тому що «Схильний до ризику» - доповнення до «обережного».

Аналогічно, для мінімаксного принципу за правилом $П_2$: $\alpha_2 = 0{,}7$ (тому що приналежність до множини обережних противників ми вже визначили).

Для мінімаксного принципу за правилом $П_3$: $\alpha_2 = min\left[\frac{1}{3,}1-\frac{25}{28}\right]$. Взяли мінімум, тому що противник повинен бути досвідченим «і» виграш повинен бути не занадто низьким, $\frac{25}{28}$, тому що «не надто низький» - доповнення до «занадто низький».

Зауважимо, що в разі не дуже низького середнього виграшу за мінімаксним принципом досвідчений противник вибере, як і в другому правилі, мінімаксне рішення. Таким чином, мінімаксне рішення приймається в одному з двох випадків, влаштовує будь-який, а значить, тут логіка «або», отже, ступінь впевненості в мінімаксному рішенні повинна бути розрахована як $\beta_2 = max[\alpha_2, \alpha_3] = max[0{,}7, \frac{3}{28}] = 0{,}7$.

Нарешті, для принципу максимізації середнього виграшу ми будемо мати:

$$\alpha_4 = min\left[\frac{1}{3}, \frac{25}{28}\right] = \frac{1}{3} \approx 0{,}33$$

Не слід дивуватися тому, що сума отриманих упевненостей для всіх правил перевищує 1, адже зазначені множини не є взаємовиключними. Наприклад, противник може бути одночасно і досвідченим, і обережним, або, навпаки, бути при цьому схильним до ризику (скоріше за все, розумного). Однак для підрахунку відповідних ймовірностей нам необхідно провести нормування.

$p_1 = \frac{\alpha_1}{\alpha_1+\beta_2+\alpha_4} = \frac{0{,}67}{0{,}67+0{,}7+0{,}33} \approx \frac{2}{5}$ - оцінка ймовірності вибору противником стратегії максимізації максимуму доходу. Аналогічно, $p_2 = \frac{\beta_2}{\alpha_1+\beta_2+\alpha_4} \approx \frac{2}{5}$ ймовірність вибору мінімаксної стратегії, $p_3 = \frac{\alpha_4}{\alpha_1+\beta_2+\alpha_4} \approx \frac{1}{5}$ - ймовірність вибору стратегії максимізації середнього виграшу.

За принципом максимізації максимуму доходів противник вибере рішення 3 або 4, тому що в цьому випадку у нього є шанс отримати 7 у.о., а при будь-якому іншому рішенні він виграє не більше 5 у.о. Тоді, кожен з цих виборів будемо вважати

рівноймовірними з $p_4=\frac{1}{5}$. Якщо він вибере рішення 3, то нашою оптимальною відповіддю буде рішення 3. Якщо він вибере стратегію 4, то нашою оптимальною відповіддю буде рішення 4 (збіг випадковий!). За мінімаксним принципом противник вибере рішення 1, нашою оптимальною відповіддю буде рішення 3. Нарешті, за принципом максимізації середнього виграшу (сума значень в стовпці, взята із зворотним знаком) противник вибере рішення 1 або рішення 3. Їх також можна вважати рівноімовірними з імовірностями $p_5=\frac{1}{10}$, але нашою оптимальною відповіддю в будь-якому випадку тут буде рішення 3.

Підсумуємо відповідні ймовірності для кожної з наших оптимальних відповідей. Рішення 3 ми повинні прийняти з ймовірністю $p_6=\frac{1}{5}+\frac{2}{5}+\frac{1}{5}=\frac{4}{5}$ і рішення 4 з ймовірністю $p_7=\frac{1}{5}$. Це і є оптимальні змішані стратегії. На практиці їх можна реалізувати, наприклад, за методом пропорційної рулетки: розбити рулетку на 5 однакових секторів, 1 зафарбувати чорним кольором, 4 - білим. Якщо стрілка вкаже на білий сектор, то вибрати рішення 3, якщо на чорний - рішення 4.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

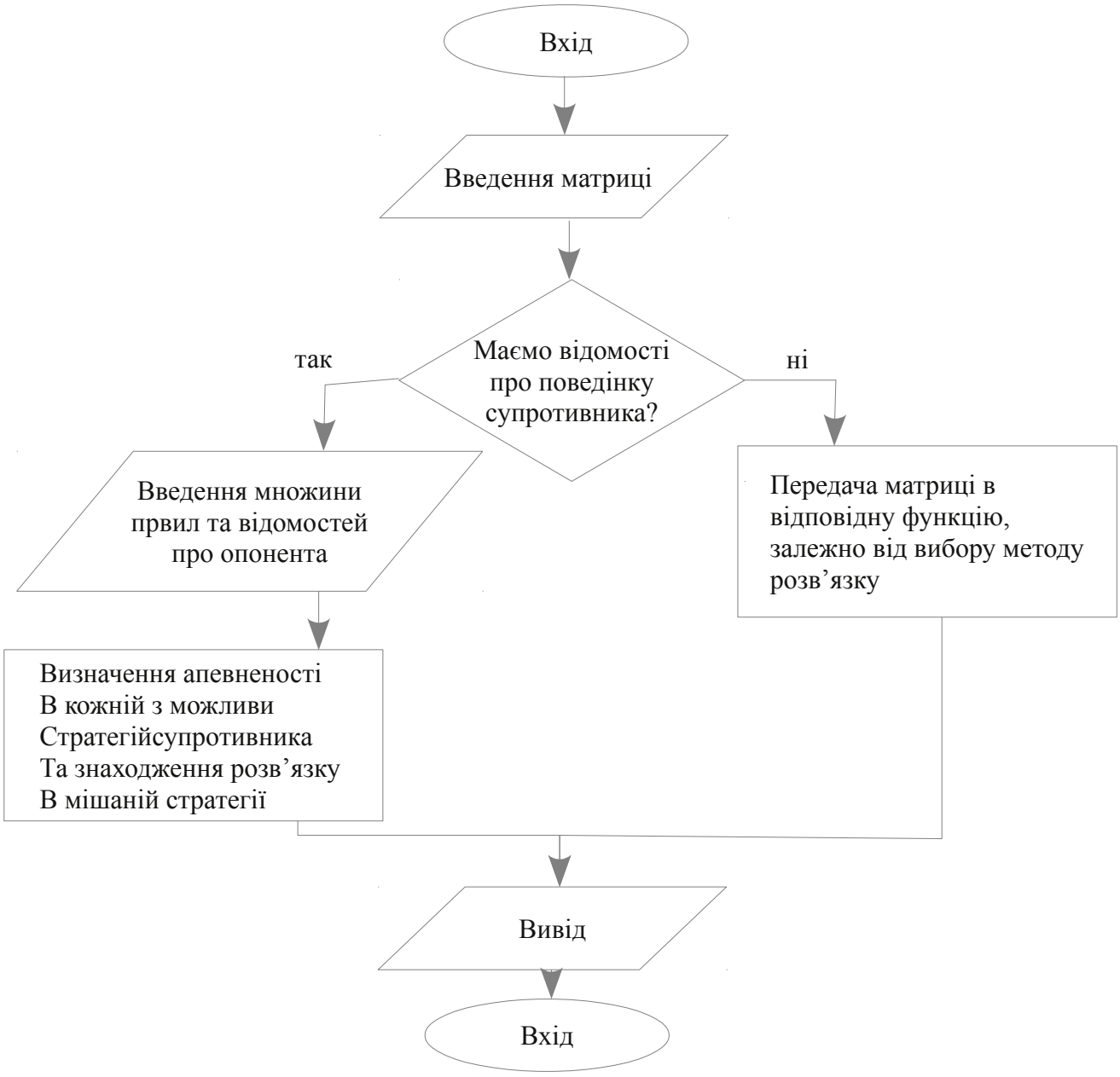
3.1 Структура та опис програмного забезпечення

Дипломна робота повністю реалізована на JavaScript. Це дає змогу розширити кількість пристроїв, які зможуть виконувати програму без потреби перекомпіляції. Поділяється на клієнтську та серверну частини.

В якості бази для проектування інтерфейсу було використано фреймворк EnyoJS. Це досить молодий проект. Використовувався в розробці інтерфейсу WebOS.

Основний код міститься в файлі myApp/myApp.js.

Алгоритм роботи програми можна описати так.



В каталозі matrixgames зберігаються файли, які містять функції, в який реалізовано різні методи розв’язання матричних ігор.

Виключенням є файл “helpers.js”, де зберігаються функції для форматованого виводу, роботи з матрицями, та інші допоміжні функції.

Файл “calculate.js” визначає потрібний метод та передає йому потрібні параметри.

Графічний метод міститься в двох файлах. Файл “Nx2xN.js” викликається на клієнті і на сервері — тут обраховується рішення. Також слієнтська частина використовує “client_paint_canvas_graph.js”, а серверна “nodejs_paint_canvas_graph.js”. Таке розділення використовується для того, що сервер використовує трохи змінену бібліотеку для “canvas”. А також не використовує затримки при малюванні. Для ефекту плавності при малюванні графіку в браузері використовується невелика затримка.

Якщо розміри матриці невеликі, то можна обрахувати розультат прямо в браузері і не використовувати інтернет-з’єднання для цього. Методи роботи з нечіткими множинами реалізовані тільки на клієнтській частині.

В випадку, кому матриця має значні розміри має сенс ввести її в текстовий файл і відіслати на сервер (матричну гру, введену в браузері теж можна порахувати на сервері). На сервері реалізовано три основних методи розв’язку матричних ігор: графічний метод, метод Брауна-Робінсона та симплекс-метод. Дані кодуються в формат JSON та передаються на сервер за протоколом WebSocket.

Примітка. При введені матричної гри в файл, немає змоги вирішити її в браузері.

Серверна частина працює під платформою nodejs. Я обрав саме цю платформу, тому що це дало змогу писати на одній мові на клієнті та на сервері. Також як сервер може використовуватися будь-який дистрибутив ОС Linux, Windows, FreeBSD та деякі інші операційні системи. До стандартного набору модулів nodejs потрібно було довстановити node-o3-canvas та socket.io.

При обрахунках як на сервері так і в браузері — формат виводу не змінюється.

3.2 Опис роботи програми та результати розрахунків

Формат файлу, що містить матричну гру.

Файл складається з рядків чисел, самі числа мають бути записані через пробіл.

Дробові числа вводяться через крапку.

Приклад.

2 3

3 1

1 5

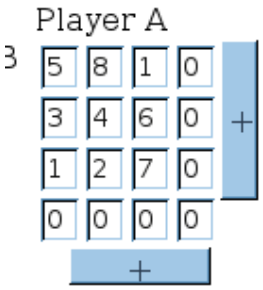
Вирішення матричної гри симплекс-методом.

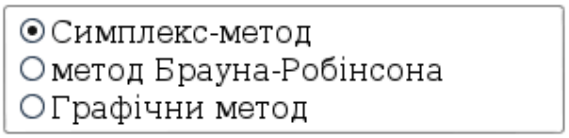
Задача.

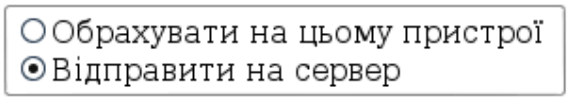
5 8 1

3 4 6

1 2 7







Рішення:

Для першого гравця:

$F = x_1 + x_2 + x_3 \rightarrow \min$

З обмеженнями:

$5x_1 + 3x_2 + 1x_3 >= 1$
$8x_1 + 4x_2 + 2x_3 >= 1$
$1x_1 + 6x_2 + 7x_3 >= 1$
$x_j >= 0.\ j = 1..3$

Для другого гравця:
$Ф = y_1 + y_2 + y_3 \rightarrow max$

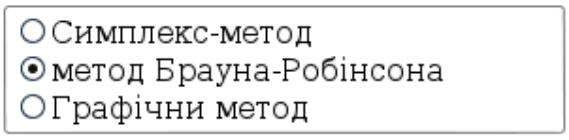
З обмеженнями:

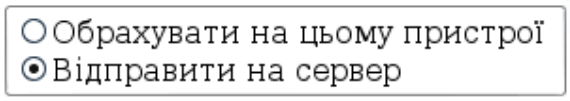
$5y_1 + 8y_2 + 1y_3 <= 1$
$3y_1 + 4y_2 + 6y_3 <= 1$
$1y_1 + 2y_2 + 7y_3 <= 1$
$y_j >= 0.\ j = 1..3$

Використавши симплекс метод

Результат:
$x^*(0.71; 0; 0.29)$
$y^*(0.43; 0.57; 0)$
Ціна гри = 3.86

Перевіримо результат, обравши метод Брауна-Робінсона.



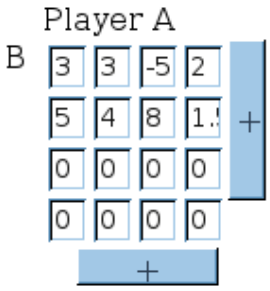


Після використання методу Брауна-Робінсона
Після 1000 ітерацій ми маємо:
$x^*(0.7; 0; 0.3)$
$y^*(0.43; 0.57; 0)$
Ціна гри = 3.88

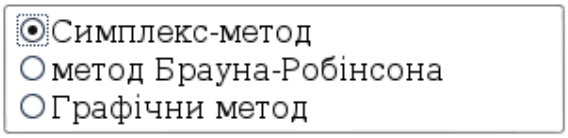
Задача розмірності 2*N

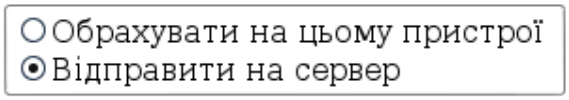
3 3 -5 2

5 4 8 1,5



Симплекс-метод





Рішення:

Для першого гравця:

$F = x_1 + x_2 \rightarrow \min$

З обмеженнями:

$8x_1 + 10x_2 >= 1$

$8x_1 + 9x_2 >= 1$

$0x_1 + 13x_2 >= 1$

$7x_1 + 6.5x_2 >= 1$

$x_j >= 0.\ j = 1..2$

Для другого гравця:

$\Phi = y_1 + y_2 \rightarrow \max$

З обмеженнями:

$8y_1 + 8y_2 + 0y_3 + 7y_4 <= 1$

$10y_1 + 9y_2 + 13y_3 + 6.5y_4 <= 1$

$y_j >= 0.\ j = 1..4$

Використавши симплекс метод

Результат:

$x^*(0;\ 0;\ 0.04;\ 0.96)$

$y^*(0.48;\ 0.52)$

Ціна гри = 11.74

Метод Брауна-Робінсона

○ Симплекс-метод
◉ метод Брауна-Робінсона
○ Графічни метод

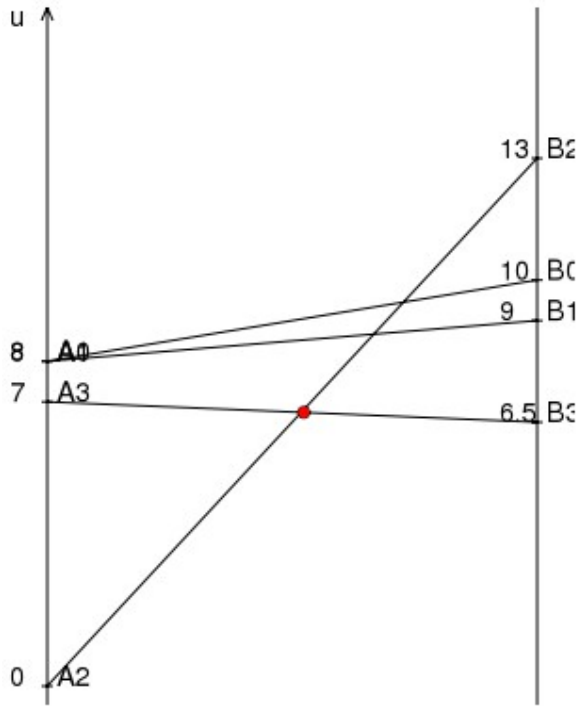
○ Обрахувати на цьому пристрої
◉ Відправити на сервер

```
Після використання методу Брауна-Робінсона
Після 1000 ітерацій ми маємо:
x*(0; 0; 0.03; 0.97)
y*(0.48; 0.52)
Ціна гри = 11.75
```

Графічний метод.

○ Симплекс-метод
○ метод Брауна-Робінсона
◉ Графічни метод

○ Обрахувати на цьому пристрої
◉ Відправити на сервер




```
x*(0; 0; 0.04; 0.96)
y*(0.48; 0.52)
Ціна гри = 11.74
```

Розв’язування гри в нечітких стратегіях

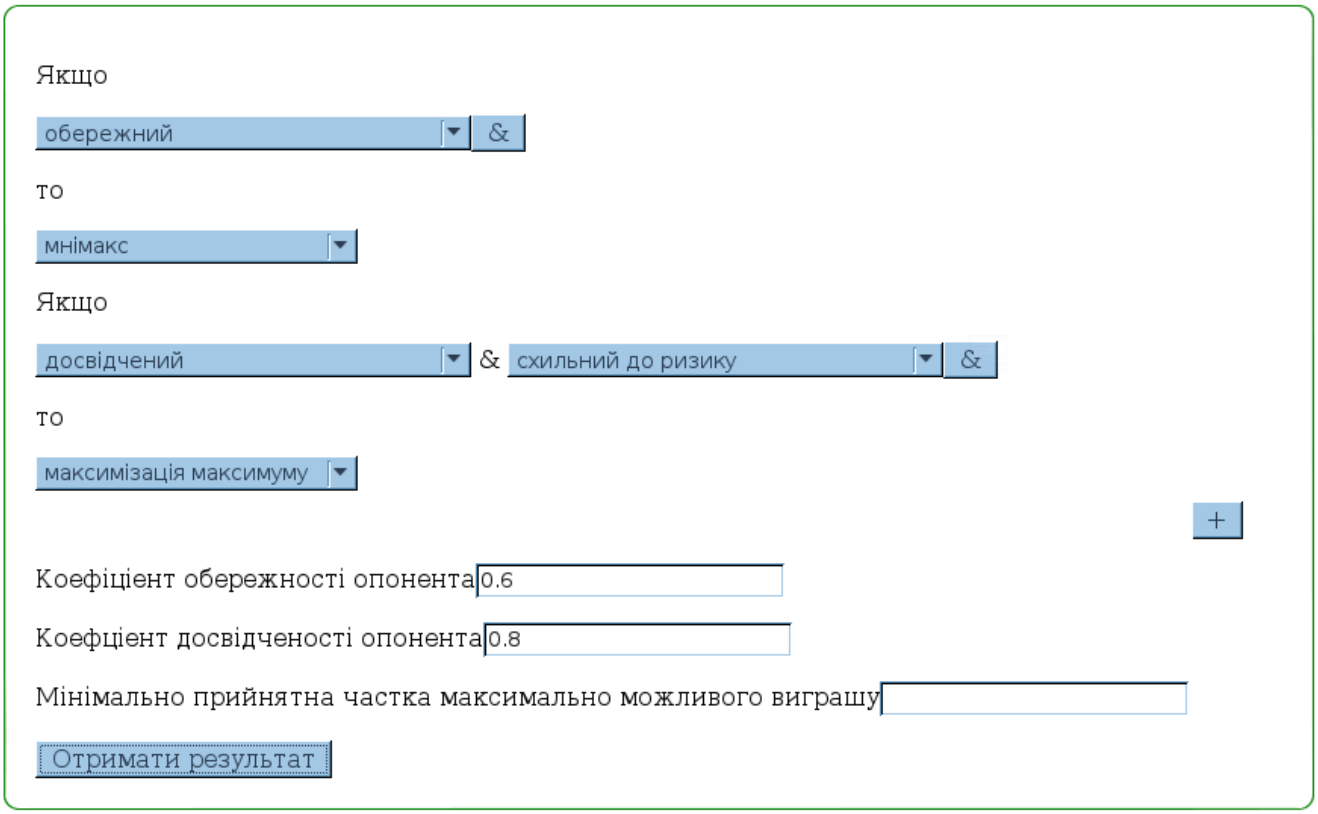
Нехай маємо матрицю

-3 8 -5
5 5 2
1 6 5

Нехай маємо базу з двох правил:

$П_1$: {Якщо супротивник не схильний до ризику, то обере стратегію мінімаксу}.

$П_2$: {Якщо супротивник досвідчений та схильний до ризику, то обере стратегію максимізації максимуму}.



Нехай вважаємо, що коефіцієнт обережності 0,6, а коефіцієнт досвіду 0,8.

Після виконання команди “Отримати результат маємо”:

x* = (0.6; 0; 0.4; 0)

Висновки

Задачі, що зводяться до матричних ігор зустрічаються в багатьох аспектах життя, особливо в економіці. Багато вчених проводили дослідження з оптимізації розв'язку матричних ігор. Особливо виділяються методи приведення до задачі лінійного програмування та ітеративні методи, такі як метод Брауна-Робінсона та монотонний ітеративний алгоритм. Прості ігри виду 2*n та m*2 можна розв’язувати графічно.

Відомий математик, лауреат нобелівської премії в галузі економіки, Джон Неш досліджував матричні ігри з ненульовою сумою, в яких існують ситуації, вигідні відразу кільком (чи всим) гравцям.

В даній дипломній роботі було реалізовано алгоритми розв’язування матричних ігор. А саме симплекс-метод, метод Брауна-Робінсона та графічний метод.

Також було реалізовано алгоритм, що при обчислені враховує прогнозування поведінки супротивника на основі попередній відомостей про нього. Цей алгоритм будує змішану стратегію, в якій враховано степінь впевненості в виборі супротивником кожної своєї стратегії.

Програма має можливість проводити складні обчислення на стороні сервера, що значно зменшує навантаження на пристрій. Це особливо актуально у випадку планшету чи мобільного телефону. Було протестовано на планшеті HP TpuchPad з операційною системою WebOS та браузером на основі WebKit та телефоні HTC Dezire Z з операційною системою Android та браузером на основі WebKit.

СПИСОК ЛІТЕРАТУРИ

1. http://nodebeginner.ru/
2. http://enyojs.com/#documentation
3. http://javascript.ru/manual
4. Зайченко Ю.П. Исследование операций.- Киев: Вища школа, 1988.- 552с.
5. Конюховский П. В. Математические методы исследования операций в экономике. — СПб: Питер, 2000. — 208 с
6. Крушевский А.В. Теория игр.- Киев: "Вища школа", 1977, 216с
7. http://math.semestr.ru/
8. Решение матричных игр в нечетких смешанных стратегиях. П.П. Петтай
9. Бесконечные антагонистические игры. — Сб. под ред. Н. Н. Воробьева. М., Физматгиз, 1963.
10. Суздаль В. Г. Теория игр для флота. М., Воениздат, 1976.

ДОДАТОК А. Вихідні коди програми

Файл client/myApp/myApp.js

```
var params = {};
var computation_place = '';
var method = '';
params['parties_count'] = 1000;
params['show_steps'] = false;

/**
 * Function : dump()
 * Arguments: The data - array,hash(associative array),object
 *    The level - OPTIONAL
 * Returns  : The textual representation of the array.
 * This function was inspired by the print_r function of PHP.
 * This will accept some data as the argument and return a
 * text that will be a more readable version of the
 * array/hash/object that is given.
 * Docs: http://www.openjs.com/scripts/others/dump_function_php_print_r.php
 */
function dump(arr,level) {
        var dumped_text = "";
        if(!level) level = 0;

        //The padding given at the beginning of the line.
        var level_padding = "";
        for(var j=0;j<level+1;j++) level_padding += "    ";

        if(typeof(arr) == 'object') { //Array/Hashes/Objects
                for(var item in arr) {
                        var value = arr[item];
```

```
                        if(typeof(value) == 'object') { //If it is an array,
                                dumped_text += level_padding + "'" + item + "' ...\n";
                                dumped_text += dump(value,level+1);
                        } else {
                                dumped_text += level_padding + "'" + item + "'
=> \"" + value + "\"\n";
                        }
                }
        } else { //Stings/Chars/Numbers etc.
                dumped_text = "===>"+arr+"<===("+typeof(arr)+")";
        }
        return dumped_text;
}

var title = new enyo.Control({
  name: "Title",
  tag: 'h1',
  content: 'Введіть умову гри'
});
title.write();

new enyo.Control({tag: 'p', content:'Стратегії для кожного гравця'}).write();

var matrix = new enyo.Control({
    name:"Matrix",
    tag: "div",
    classes : 'strategies',
    components: [
        { tag: "div", classes: "blankblock", content: ' ' },
        { tag: "div", classes: "label1", content: 'Player A' },
        { tag: "div", classes: "label2", content: 'B' },
```

```
        { name:'matrixTable', tag: "table", components: [
            {tag: 'tr', components: [
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]}
            ]},
            {tag: 'tr', components: [
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]}
            ]},
            {tag: 'tr', components: [
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]}
            ]},
            {tag: 'tr', components: [
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]},
                {tag: 'td', components: [{tag: 'input'}]}
            ]}
        ]} ,
        { name: 'addV', tag: "button", classes:'addV', content: "+" },
        { name: 'addH', tag: "button", classes:'addH', content: "+" },
    ]
});
```

```
var fileuploader = new enyo.Control({
    name : 'upload',
    components: [
        {tag: 'input', id: 'fileupload', attributes: { type: "file", multiple: 'false' }},
        {tag: 'p', attributes: {id: 'info'}, content: 'Перетяніть файл на це поле для його завантаження.<br>Або натисніть сюди для вибору файла.'}
    ]
});

new enyo.Control({
    name: 'inputs',
    components: [
        matrix,
        {tag: 'span', content: 'або виберіть файл'},
        fileuploader
    ]
}).write();

new enyo.Control({
    name: 'params',
    components: [
        {name:"method",
        tag: "div",
        components: [
            { components:[
                { tag: 'input type="radio" name="method" value="symplex" checked="checked"' },
                { tag: 'label', content: 'Симплекс-метод' }
            ]},
            { components:[
                {tag: 'input type="radio" name="method" value="br"' },
```

```
                { tag: 'label', content: 'метод Брауна-Робінсона' }
            ]},
            { components:[
                {tag: 'input type="radio" name="method" value="graph"' },
                { tag: 'label', content: 'Графічни метод' }
            ]}
        ]},
        {name:"where",
        tag: "div",
        components: [
            { components:[
                { tag: 'input type="radio" name="where" value="device"
checked="checked"' },
                { tag: 'label', content: 'Обрахувати на цьому пристрої' }
            ]},
            { components:[
                {tag: 'input type="radio" name="where" value="server"' },
                { tag: 'label', content: 'Відправити на сервер' }
            ]}
        ]}
    ]
}).write();

goButton = new enyo.Control({
    tag: 'Button',
    name: 'GoButton',
    classes: 'GoButton',
    content: 'Знайти розв’язок'
});

goButton.write();
```

```
        resultArea = new enyo.Control({
            tag: 'div',
            name: 'resultArea',
            classes: 'resultArea',
            content:' '
        });

        predekat = new enyo.Control({
            name: 'predekat',
            classes: 'predekat',
            components: [
                {tag: "select",
                    components: [
                        {tag: 'option value="experienced"', content: "досвідчений"},
                        {tag: 'option value="inexperienced"',content: "недосвідчений"},
                        {tag: 'option value="at risk"',content: "схильний до ризику"},
                        {tag: 'option value="cautious"',content: "обережний"},
                        {tag: 'option value="gain is too low"',content: "середній виграш
надто низький"},
                        {tag: 'option value="gain is not too low"',content: "середній
виграш не надто низький"},
                    ]
                }
            ]
        });

        strategy = new enyo.Control({
            name: 'strategy',
            classes: 'strategy',
            components: [
```

```
            {tag: "select", components: [
                {tag: 'option value="maximizing the maximum"', content:
"максимізація максимуму"},
                {tag: 'option value="minimax"', content: "мнімакс"},
                {tag: 'option value="maximizing the average"', content:
"максимізація середнього"}
            ]}
        ]
    });

    rule = new enyo.Control({
        name: 'rule',
        classes: 'rule',
        components: [
            {
                tag: 'p',
                content: 'Якщо'
            },
            predekat,
            {
                tag: 'button',
                name: 'and',
                content:'&amp;amp;',
                classes: 'rule_and'
            },
            {
                tag: 'p',
                content: 'то'
            },
            strategy
        ]
```

```
});

opponent_info = new enyo.Control({
    name: 'opponent_info',
    components: [
        {
            tag: 'p',
            components: [
                {
                    tag: 'label',
                    content: 'Коефіцієнт обережності опонента'
                },{
                    tag: 'input',
                    name: 'caution',
                    value: '0'
                }
            ]
        },
        {
            tag: 'p',
            components: [
                {
                    tag: 'label',
                    content: 'Коефцієнт досвіду опонента'
                },{
                    tag: 'input',
                    name: 'experience',
                    value: '0'
                }
            ]
        },
```

```
            {
               tag: 'p',
               components: [
                  {
                     tag: 'label',
                     content: 'Мінімально прийнятна частка максимально
можливого виграшу'
                  },{
                     tag: 'input',
                     name: 'low_gain',
                     value: '0'
                  }
               ]
            }
         ]
      });

      rules = new enyo.Control({
         name: 'rules',
         components: [
            rule,
            {
               tag: 'button',
               name: 'plus',
               content:'+'
            },
            opponent_info,
            {
               tag: 'button',
               name: 'get_result',
               content:'Отримати результат'
```

```
            },
        ]
    });

    use_opponent_information = new enyo.Control({
        name: 'use_opponent_information',
        tag: 'button',
        content: 'Додаткові відомості про опонента'
    }).write();

    rules.write();
    resultArea.write();

    function writeResult(result, method){
        var output = '<pre>';
        if(result['output'] != undefined)
            output += result['output'];
        if(method=='br'){
            output += 'Після використання методу Брауна-Робінсона' + "\n";
            output += 'Після ' + params['parties_count'] + ' ітерацій ми маємо:' +
"\n";
        }else if(method=='symplex'){
            output += 'Використавши симплекс метод' + "\n";
            output += "\n"+'Результат:' + "\n";
        }
        if(result['player1_strategies'] != undefined)
            output += 'x<sup>*</sup>(' + result['player1_strategies'].join('; ')+
')'+"\n";
        if(result['player2_strategies'] != undefined)
            output += 'y<sup>*</sup>(' + result['player2_strategies'].join('; ')+
')'+"\n";
```

```
            if(result['Game_price'] != undefined)
                output += 'Ціна гри = ' + (result['Game_price']-result['maxNegative']) +
"\n";

            if(!params['show_steps'])
                $('#resultArea').html(output+'</pre>');
        }

        function inialise_rule_and(){
            $('.rule_and').click(function(){
                $(this).parent().children('button').before("<span> & </span>");
                $(this).parent().children('button').before(predekat.generateHtml());
            });
        }

        $(document).ready(function(){

           $('#use_opponent_information').click(function(){$('#rules').show(1000)});

            inialise_rule_and();

            $('#rules_plus').click(function(){
                $(this).parent().children('button').first().before(rule.generateHtml());

                inialise_rule_and();
            });

            $('#rules_get_result').click(function(){
                var mu = {};
                mu['A'] = $('#opponent_info_experience').val();
                mu['B'] = $('#opponent_info_caution').val();
```

```
var matrix = new Array();

$('#Matrix_matrixTable tr').each(function(){
    var arr = Array();
    $(this).children('td').each(function(){
        if($(this).children('input').val()=='')
            $(this).children('input').val(0);
        arr.push(parseFloat($(this).children('input').val()));
    });
    matrix.push(arr);
});

var zmax = undefined;

for(j=0;j<matrix[0].length;j++){
    for(i=0;i<matrix.length;i++){
        if(matrix[i][j]*(-1)>zmax || zmax == undefined)
            zmax = matrix[i][j]*(-1);
    }
}

var minimax = undefined;
var minimaxCols = [];

var maximax = undefined;
var maximaxCol = [];

var maxiavg = undefined;
var maxiavgCol = [];
```

```
for(j=0;j<matrix[0].length;j++){
    min = undefined;
    for(i=0;i<matrix.length;i++){
        if(matrix[i][j]*(-1)<min || min == undefined)
            min = matrix[i][j]*(-1);
    }

    if(minimax<min || minimax == undefined){
        minimax = min;
        minimaxCols = [j];
    }else if(minimax == min){
        minimaxCols.push(j);
    }
}

for(j=0;j<matrix[0].length;j++){
    max = undefined;
    for(i=0;i<matrix.length;i++){
        if(matrix[i][j]*(-1)>max || max == undefined)
            max = matrix[i][j]*(-1);
    }

    if(maximax<max || maximax == undefined){
        maximax = max;
        maximaxCols = [j];
    }else if(maximax == max){
        maximaxCols.push(j);
    }
}

for(j=0;j<matrix[0].length;j++){
```

```
            sum = 0;
            for(i=0;i<matrix.length;i++){
                sum += matrix[i][j]*(-1);
            }

            if(maxiavg<sum || maxiavg == undefined){
                maxiavg = sum;
                maxiavgCols = [j];
            }else if(maxiavg == sum){
                maxiavgCols.push(j);
            }
        }

        var z_minimax_sum = 0;

        for(j=0;j<minimaxCols.length;j++)
            for(i=0;i<matrix.length;i++){
                z_minimax_sum += matrix[i][minimaxCols[j]]*(-1);
            }

        var z_minimax_avg = z_minimax_sum /
(matrix.length*minimaxCols.length);

        var low_gain = $('#opponent_info_low_gain').val();

        if(z_minimax_avg <= 0) mu['D'] = 0;
        else if(z_minimax_avg > 0 && z_minimax_avg <low_gain*zmax){
            mu['D'] = (low_gain*zmax - z_minimax_avg)/(low_gain*zmax);
        }else{
            mu['D'] = 0;
        }
```

```
var strategies = {
    'maximizing the maximum' : 0,
    'minimax'                : 0,
    'maximizing the average' : 0
};

for (s in strategies){
    $('.rule').each(function(){
        if($(this).find('.strategy option:selected').val() == s){

            tmp = undefined;

            $(this).find('.predekat option:selected').each(function(){
                predekat = $(this).val();

                val = undefined;

                if(predekat=='experienced'){
                    val = mu['A'];
                }else if(predekat=='inexperienced'){
                    val = 1 - mu['A'];
                }else if(predekat=='at risk'){
                    val = 1 - mu['B'];
                }else if(predekat=='cautious'){
                    val = mu['B'];
                }else if(predekat=='gain is too low'){
                    val = mu['D'];
                }else if(predekat=='gain is not too low'){
                    val = 1 - mu['D'];
                }
```

```
            if(tmp == undefined || tmp > val){
                tmp = val;
            }

        });

        if(strategies[s] < tmp){
          strategies[s] = tmp;
        }
      }
   });
}

sum = 0;
for (s in strategies){
   sum += parseFloat(strategies[s]);
}

for (s in strategies){
   strategies[s] = strategies[s] / sum;
}

result = Array(matrix.length);
for(i = 0;i<matrix.length;i++){
   result[i] = 0;
}

for(i in maximaxCols){

   max = undefined;
```

```
            otvet = 0;
            for(j=0;j<matrix.length;j++){
                if(max == undefined || matrix[j][maximaxCols[i]] > max){
                    max = matrix[j][maximaxCols[i]];
                    otvet = j;
                }
            }

            result[otvet] += strategies['maximizing the maximum'] /
maximaxCols.length;
        }
        for(i in minimaxCols){

            max = undefined;
            otvet = 0;
            for(j=0;j<matrix.length;j++){
                if(max == undefined || matrix[j][minimaxCols[i]] > max){
                    max = matrix[j][minimaxCols[i]];
                    otvet = j;
                }
            }

            result[otvet] += strategies['minimax'] / minimaxCols.length;
        }
        for(i in maxiavgCols){

            max = undefined;
            otvet = 0;
            for(j=0;j<matrix.length;j++){
                if(max == undefined || matrix[j][maxiavgCols[i]] > max){
                    max = matrix[j][maxiavgCols[i]];
```

```
                    otvet = j;
                }
            }

            result[otvet] += strategies['maximizing the average'] /
maxiavgCols.length;
        }

        result['output'] = 'x* = (' + result.map(function(x){return
Math.round((x)*100)/100;}).join('; ') + ')';

        writeResult(result, '');

    });

    $('#params_where input[type="radio"]').change(function()
{computation_place = $(this).val();});
    $('#params_method input[type="radio"]').change(function(){method = $
(this).val();});

    computation_place = $('#params_where
input[type="radio"]:checked').val();
    method = $('#params_method input[type="radio"]:checked').val();

    var socket = io.connect('http://matrixgames.org.ua:8888');

    socket.on('connect', function() {
        console.log('connected');
    });

    socket.on('message', function(data) {
```

```
            alert(data);
        });

        socket.on('result', function(data) {
            result = JSON.parse(data);writeResult(result, method);
        });



        if ($.browser.msie || $.browser.opera) {
            $(document.body).text('Ваш браузер не підтримує передачу файлів через Drag&Drop.');
            return;
        }

        function handleDrag(e) {
            if (e.type == 'dragenter') {
              $('#upload').addClass('drop');
            } else if ((e.type == 'dragleave') || (e.type == 'drop')) {
              $('#upload').removeClass('drop');
            }
            e.stopPropagation();
            e.preventDefault();
        }

        function handleUploads(files) {

            if(computation_place == 'device'){
                alert('Завантажувати файл можна тільки на сервер');return false;
            }

            for (var i = 0; i < files.length; ++i) {
```

```
                var reader = new FileReader();
                reader.onloadend = function(d) {
                    socket.send(JSON.stringify({matrix:d.target.result, method: method,
params: JSON.stringify(params)}), function(err) {
                        if (!err) {
                            console.log('file uploaded');
                        }
                    });
                };
                reader.readAsDataURL(files[i]);
            }
        }

        $('#upload, #fileupload').bind('dragenter', handleDrag).bind('dragleave',
handleDrag).bind('dragover', handleDrag);

        $('#upload').get(0).ondrop = function(e) {
            handleDrag(e);
            if (!e.dataTransfer.files) {
                alert('Ваш браузер не підтримує передачу файлів через
Drag&Drop.');
                return;
            }
            var files = e.dataTransfer.files;
            handleUploads(files);
        };

        $('#fileupload').change(function(e) {
            handleUploads(this.files);
        }).click(function(e) {
            e.stopPropagation();
```

```
        });

        if ($.browser.mozilla) {
            $('#upload').click(function() {
                $('#fileupload').click();
            });
            $('#fileupload').css('display', 'none');
        }

        $('#Matrix_addV').click(function(){
            $('#Matrix').width($('#Matrix').width()+24);
            $('#Matrix_matrixTable tr').each(function(){$
(this).append('<td><input /></td>');});
        });
        $('#Matrix_addH').click(function(){
            $('#Matrix').height($('#Matrix').height()+24);
            $('#Matrix_matrixTable').append($("#Matrix_matrixTable
tr:first").clone());
        });
        $('#GoButton').click(function(){

            var matrix = new Array();

            $('#Matrix_matrixTable tr').each(function(){
                var arr = Array();
                $(this).children('td').each(function(){
                    if($(this).children('input').val()=='')
                        $(this).children('input').val(0);
                    arr.push(parseFloat($(this).children('input').val()));
                });
                matrix.push(arr);
```

```
            });

            if(computation_place == 'device'){

                var result = calculate(matrix, method, params);

                if(method == "graph"){
                    client_paint_canvas_graph(result['max'], result['min'],
result['maxX'], result['targY'], matrix, result['n2']);
                }

                writeResult(result, method);
            }else if(computation_place == 'server'){
                socket.emit('calculate', { 'matrix': JSON.stringify(matrix), 'method':
method,  params: JSON.stringify(params) });
            }
        });
    });
```

Файл client/matrixgames/BraunRobinson.js

```
    function BraunRobinson(matrix, params){

        var player1_strategy = undefined;
        var player2_strategy = undefined;
        var u = undefined;
        var w = undefined;

        var parties_count = params['parties_count'];

        var result = {};
```

```
        result['player1_strategies'] = Array(matrix[0].length);
            for(i=0;i<result['player1_strategies'].length;i+
+)result['player1_strategies'][i] = 0;
        result['player2_strategies'] = Array(matrix.length);
            for(i=0;i<result['player2_strategies'].length;i+
+)result['player2_strategies'][i] = 0;

        var player1_sums = Array(matrix.length);
            for(i=0;i<player1_sums.length;i++)player1_sums[i] = 0;
        var player2_sums = Array(matrix[0].length);
            for(i=0;i<player2_sums.length;i++)player2_sums[i] = 0;

        var part_number;
        for(part_number=0;part_number<parties_count;part_number++){

            var max = 0;
            if(player1_strategy == undefined){
                for(i=0;i<matrix.length;i++)
                    for(j=0;j<matrix[i].length;j++)
                        if(matrix[i][j]>max){
                            player2_strategy = i;
                            max = matrix[i][j];
                        }
                player1_strategy = 0;
            }else{
                for(i=0;i<player1_sums.length;i++)
                    if(player1_sums[i]>player1_sums[player2_strategy]){
                        player2_strategy = i;
                    }
            }
```

```
            result['player2_strategies'][player2_strategy]++;
            for(i=0;i<matrix[player2_strategy].length;i++)
                player2_sums[i]+=matrix[player2_strategy][i];

            for(i=0;i<player2_sums.length;i++){
                if(player2_sums[i]<player2_sums[player1_strategy]){
                    player1_strategy = i;
                }
            }

            result['player1_strategies'][player1_strategy]++;

            for(i=0;i<matrix.length;i++)
                player1_sums[i]+=matrix[i][player1_strategy];
        }

        result['player1_strategies'] = result['player1_strategies'].map(function(x)
{return Math.round((x/parties_count)*100)/100;});
        result['player2_strategies'] = result['player2_strategies'].map(function(x)
{return Math.round((x/parties_count)*100)/100;});
        result['Game_price'] = (Math.round(((Math.min.apply(null, player2_sums) +
Math.max.apply(null, player1_sums))/(parties_count*2))*100)/100);

        return result;
    }
```

Файл client/matrixgames/calculate.js

```
    function calculate(matrix, method, params){

        var result = {};
```

```
    /* Deleting null-lines */
    matrix = delete_null_lines(matrix);

    /* find dominant colls and rows */
    //matrix = find_dominant_colls_and_rows(matrix);

    /* Чи знайдено сідлову точку */
    if(matrix.length == 1 && matrix[0].length == 1){
        result['Game_price'] = matrix[0][0];
    }

    /* remove the negative items */
    result = remove_the_negative_items(matrix);
    matrix = result['matrix'];
    var maxNegative = result['maxNegative'];

    var basis = Array(matrix.length);
    if(method=='symplex'){
        result = symplex(matrix, params);
    }else
    if(method=='br'){
        result = BraunRobinson(matrix, params);
    }else
    if(method == "graph"){
        result = Nx2xN(matrix);
    }
    result['maxNegative'] = maxNegative;
    return result;
}
```

Файл client/matrixgames/client_paint_canvas_graph.js

```
        function client_paint_canvas_graph(max, min, maxX, targY, matrix, n2){
            /* Check for 2*n or n*2 */
            if((matrix.length == 2 && matrix[0].length > 1) || (matrix[0].length == 2
&& matrix.length>1)){

                max    = parseFloat(max);
                min    = parseFloat(min);
                maxX   = parseFloat(maxX);
                targY  = parseFloat(targY);

                $.fancybox('<canvas id="graph" width="300"
height="400"></canvas>');

                var canvas = document.getElementById('graph');
                var ctx = canvas.getContext('2d');
                if (canvas.getContext){

                    ctx.fillStyle="#000";

                    ctx.moveTo(20,20);
                    ctx.lineTo(20,390);
                    setTimeout(function() {ctx.stroke();},500);
                    ctx.moveTo(20,20);
                    ctx.lineTo(17,27);
                    ctx.moveTo(20,20);
                    ctx.lineTo(23,27);
                    setTimeout(function() {ctx.stroke();},1000);
                    ctx.font = "15px Arial";
                    ctx.fillText("u", 0, 30);
                    setTimeout(function() {ctx.stroke();},1500);
```

```
                ctx.moveTo(280,20);
                ctx.lineTo(280,390);
                setTimeout(function() {ctx.stroke();},2000);

                var price = 280/(max-min);
                if(max>0 && min<0){
                    ctx.moveTo(0,  380+min*price);
                    ctx.lineTo(300,380+min*price);
                    setTimeout(function() {ctx.stroke();},2500);
                    ctx.moveTo(300,380+min*price);
                    ctx.lineTo(287,380+min*price-3);
                    ctx.moveTo(300,380+min*price);
                    ctx.lineTo(287,380+min*price+3);
                    setTimeout(function() {ctx.stroke();},3000);
                }

                for(i=0;i<2;i++)
                    for(j=0;j<matrix[i].length;j++){
                        ctx.moveTo(20+i*260-3,380+(-matrix[i][j]+min)*price);
                        ctx.lineTo(20+i*260+3,380+(-matrix[i][j]+min)*price);
                        ctx.fillText(matrix[i][j], 20+i*260-20,380+(-matrix[i][j]
+min)*price);
                        ctx.fillText((i==0?'A':'B')+j, i*260+(i==20?0:25),380+(-matrix[i]
[j]+min)*price);
                        setTimeout(function() {ctx.stroke();},3500+500*i);
                    }

                if(matrix[0].length == 2 && matrix.length != 2)
                    matrix = transpose(matrix);

                for(i=0;i<matrix[0].length;i++){
```

```
                ctx.moveTo(20,380+(-matrix[0][i]+min)*price);
                ctx.lineTo(280,380+(-matrix[1][i]+min)*price);
                setTimeout(function() {ctx.stroke();},4000+500*i);
            }

            ctx.fillStyle="#ff0000";

            ctx.moveTo(280*maxX+11, 380+(-targY+min)*price);
            ctx.arc(280*maxX+11, 380+(-targY+min)*price, 3, 0, 2*Math.PI,
false);
            setTimeout(function() {ctx.fill();},4500);
        }
    }
}
```

Файл client/matrixgames/helpers.js

```
function clone(x)
{
    if (x.clone)
        return x.clone();
    if (x.constructor == Array)
    {
        var r = [];
        for (var i=0,n=x.length; i<n; i++)
            r.push(clone(x[i]));
        return r;
    }
    return x;
}
```

```
function delete_null_lines(matrix){
    for(i=matrix.length-1;i>=0;i--){
        free = true;
        for(j=0;j<matrix[i].length;j++)
            if(matrix[i][j] != 0) free = false;
        if(free)
            matrix.splice(i, 1);
    }

    for(i=matrix[0].length-1;i>=0;i--){
        free = true;
        for(j=0;j<matrix.length;j++)
            if(matrix[j][i] != 0) free = false;
        if(free)
            for(j=0;j<matrix.length;j++)
                matrix[j].splice(i, 1);
    }

    return matrix;
}

function find_dominant_colls_and_rows(matrix){
    for(i=0;i<matrix.length;i++){
        for(k=0;k<matrix.length;k++){
            if(i!=k){
                dominant = true;
                for(j=0;j<matrix[i].length;j++){
                    if(matrix[i][j] > matrix[k][j]){
                        dominant = false;
                        break;
                    }
```

```
            }
            if(dominant){
                matrix.splice(k, 1);
                k = k - 1;
                if(i>k)i--;
            }
        }
    }
}
for(i=0;i<matrix[0].length;i++){
    for(k=0;k<matrix[0].length;k++){
        if(i!=k){
            dominant = true;
            for(j=0;j<matrix.length;j++){
                if(matrix[j][i] > matrix[j][k]){
                    dominant = false;
                    break;
                }
            }
            if(dominant){
                for(j=0;j<matrix.length;j++){
                    matrix[j].splice(k, 1);
                }
                k = k - 1;
                if(i>k)i--;
            }
        }
    }
}
return matrix;
}
```

```
function remove_the_negative_items(matrix){
    var result = Array();
    var maxNegative = 0;
    for(i=0;i<matrix.length;i++){
        for(j=0;j<matrix[i].length;j++){
            if(matrix[i][j]<0 && matrix[i][j]<maxNegative)
                maxNegative = matrix[i][j];
        }
    }

    if(maxNegative<0){
        var summand = -maxNegative;
        for(i=0;i<matrix.length;i++){
            for(j=0;j<matrix[i].length;j++){
                matrix[i][j]+= summand;
            }
        }
    }
    result['matrix'] = matrix;
    result['maxNegative'] = maxNegative;
    return result;
}

/**
 * Transposes a given array.
 * @id Array.prototype.transpose
 * @author Shamasis Bhattacharya
 *
 * @type Array
 * @return The Transposed Array
```

```
 * @compat=ALL
 */
function transpose(a) {

  // Calculate the width and height of the Array
    w = a.length ? a.length : 0,
    h = a[0] instanceof Array ? a[0].length : 0;

  // In case it is a zero matrix, no transpose routine needed.
  if(h === 0 || w === 0) { return []; }

  /**
   * @var {Number} i Counter
   * @var {Number} j Counter
   * @var {Array} t Transposed data is stored in this array.
   */
  var i, j, t = [];

  // Loop through every item in the outer array (height)
  for(i=0; i<h; i++) {

    // Insert a new row (array)
    t[i] = [];

    // Loop through every item per item in outer array (width)
    for(j=0; j<w; j++) {

      // Save transposed data.
      t[i][j] = a[j][i];
    }
  }
```

```
  return t;
};
```

Файл client/matrixgames/Nx2xN.js

```
function Nx2xN(matrix){
    var result = {};
    result['output'] = '';

    /* Check for 2*n or n*2 */
    if((matrix.length == 2 && matrix[0].length > 1) || (matrix[0].length == 2
&& matrix.length>1)){
        var n2 = false;
        if(matrix[0].length == 2 && matrix.length != 2){
            n2 = true;
            matrix = transpose(matrix);
        }

        result['min'] = (Math.min.apply(null, matrix[0].concat(matrix[1])));
        result['max'] = (Math.max.apply(null, matrix[0].concat(matrix[1])));

        result['maxX'];
        if(!n2){
            result['targY'] = result['min']-1;
        }else{
            result['targY'] = result['max']+1;
        }

        var ActiveStrategy1, ActiveStrategy2;

        /* find all points of intersection */
```

```
            for(i=0;i<matrix[0].length;i++){
                for(j=0;j<matrix[0].length;j++){
                    if(i!=j){
                        /* not parallel */
                        if(matrix[0][i]-matrix[0][j] != matrix[1][i]-matrix[1][j]){
                            var x1 = 0;
                            var x2 = 1;
                            var x3 = 0;
                            var x4 = 1;

                            var y1 = matrix[0][i];
                            var y2 = matrix[1][i];
                            var y3 = matrix[0][j];
                            var y4 = matrix[1][j];

                            var x = ((x1*y2-x2*y1)*(x4-x3)-(x3*y4-x4*y3)*(x2-x1))/((y1-
y2)*(x4-x3)-(y3-y4)*(x2-x1));
                            var y = ((y3-y4)*x-(x3*y4-x4*y3))/(x4-x3);

                            if(!n2){
                                /* in minimals */
                                var cond = true;
                                for(k=0;k<matrix[0].length;k++){
                                    var x1 = 0;
                                    var x2 = 1;

                                    var y1 = matrix[0][k];
                                    var y2 = matrix[1][k];

                                    if(((y1-y2)*x-(x1*y2-x2*y1))/(x2-x1) < y || -x<0 || -x>1)
                                        cond = false;
```

```
                        }
                    }else{
                        /* in maximals */
                        var cond = true;
                        for(k=0;k<matrix[0].length;k++){
                            var x1 = 0;
                            var x2 = 1;

                            var y1 = matrix[0][k];
                            var y2 = matrix[1][k];

                            if(((y1-y2)*x-(x1*y2-x2*y1))/(x2-x1) > y || -x<0 || -x>1)
                                cond = false;
                        }
                    }
                    if(cond && ((y<result['targY'] && n2) || (y>result['targY'] && !
n2))){
                        result['targY'] = y;
                        result['maxX'] = -x;
                        ActiveStrategy1 = i;
                        ActiveStrategy2 = j;
                    }
                }
            }
        }
    }

    result['player1_strategies'] = Array();
    result['player1_strategies'].push(Math.round((1-
result['maxX'])*100)/100);
    result['player1_strategies'].push(Math.round(result['maxX']*100)/100);
```

```
            result['Game_price'] = Math.round(result['targY']*100)/100;

            result['player2_strategies'] = Array(matrix[0].length);
            for(i=0;i<matrix[0].length;i++)
                result['player2_strategies'][i] = 0;


            delta  = matrix[0][ActiveStrategy1]*matrix[1][ActiveStrategy2]-matrix[0]
[ActiveStrategy2]*matrix[1][ActiveStrategy1];
            deltax = result['Game_price']*matrix[1][ActiveStrategy2]-matrix[0]
[ActiveStrategy2]*result['Game_price'];
            deltay = result['Game_price']*matrix[0][ActiveStrategy1]-matrix[1]
[ActiveStrategy1]*result['Game_price'];

            result['player2_strategies'][ActiveStrategy2] =
Math.round(deltay*100/delta)/100;
            result['player2_strategies'][ActiveStrategy1] =
Math.round(deltax*100/delta)/100;;

            /*if(maxNegative<0){
                result['Game_price'] -= summand;
            }*/

            if(!n2){
                var tmp = clone(result['player1_strategies']);
                result['player1_strategies'] = result['player2_strategies'];
                result['player2_strategies'] = tmp;
            }else{
                result['maxX'] = result['player1_strategies'][1];
            }
        }else{
```

```
        result['output'] += 'Система не має вигляд 2*n чи n*2' + "\n";
    }

    return result;
}
```

Файл client/matrixgames/symplex.js

```
function symplex(matrix, params){

    var result = {};

    result['output'] = '';
    result['output'] += 'Рішення:' + "\n";
    result['output'] += "\n";
    result['output'] += 'Для першого гравця:' + "\n";
    result['output'] += 'F = ';
    for(i=0;i<matrix.length;i++)
        result['output'] += (i>0?'+':'')+' x<sub>'+(i+1)+'</sub> ';
    result['output'] += ' → min' + "\n";
    result['output'] += "\n";
    result['output'] += 'З обмеженнями:' + "\n";
    result['output'] += "\n";
    for(i=0;i<matrix[0].length;i++){
        for(j=0;j<matrix.length;j++)
            result['output'] += matrix[j][i]+'x<sub>'+(j+1)+'</sub>' +
((j+1)<matrix.length?" + ":'');
        result['output'] += ' >= 1' + "\n";
    }
    result['output'] += 'x<sub>j</sub> >= 0. j = 1..' + j + "\n";
    result['output'] += "\n";
    result['output'] += 'Для другого гравця:' + "\n";
```

```
            result['output'] += 'Φ = ';
            for(i=0;i<matrix.length;i++)
                result['output'] += (i>0?'+':'')+' y<sub>'+(i+1)+'</sub> ';
            result['output'] += '→ max' + "\n";
            result['output'] += "\n";
            result['output'] += 'З обмеженнями:' + "\n";
            result['output'] += "\n";
            for(i=0;i<matrix.length;i++){
                for(j=0;j<matrix[i].length;j++)
                    result['output'] += matrix[i][j]+'y<sub>'+(j+1)+'</sub>' +
((j+1)<matrix[i].length?" + ":'');
                result['output'] += ' <= 1' + "\n";
            }
            result['output'] += 'y<sub>j</sub> >= 0. j = 1..' + j + "\n";
            result['output'] += "\n";

            //Приведем систему ограничений к системе неравенств смысла ≤,
умножив соответствующие строки на (-1).
            for(i=0;i<matrix.length;i++)
                for(j=0;j<matrix[i].length;j++)
                    matrix[i][j] *= -1;
            //Transpose (for dual simplex method)
            matrix = transpose(matrix);

            var basis = Array(matrix.length);
            var F = Array(matrix.length);
            var l = matrix.length;

            var significant_elements = matrix[0].length;
            for(i=0;i<l;i++)
                for(j=0;j<l;j++){
```

```
            if(j==i){
                matrix[i].push(1);
                basis[i] = significant_elements+j;
            }else
                matrix[i].push(0);
            if(j==(l-1))
                matrix[i].push(-1);
        }
    var arr = Array(matrix[0].length);
    for(i=0;i<matrix[0].length;i++)
        if(i<significant_elements)arr[i] = -1;
        else arr[i] = 0;
    matrix.push(arr);

    function is_FcoefPositive(Arr){
        for(i=0;i<Arr.length;i++){
            if(Arr[i][Arr[i].length-1]<0) return false;
        }
        return true;
    }

    if(params['show_steps'] == 1){
        var str = "<table border=1>";

        for(i=0;i<matrix.length;i++){
            str += "<tr>";
            str += "<td>"+basis[i]+"</td>";
            str += "<td>";
            str += matrix[i].map(function(x){return
Math.round(x*100)/100;}).join('</td><td>');
            str += "</td></tr>";
```

```
        }
        str += "</table>";
        $("#resultArea").html($("#resultArea").html()+str+"<br />");
    }

    while(!is_FcoefPositive(matrix)){

        var max    = undefined; // minimal coefficient
        var minRow = undefined; // column to search minimal division

        for(i=0;i<matrix.length-1;i++)
            if(matrix[i][matrix[i].length-1]<0){
                if(matrix[i][matrix[i].length-1] < 0){
                    is_neg = false;
                    for(j=0;j<matrix[i].length-1;j++){
                        if(matrix[i][j]<0){
                            is_neg = true;
                            break;
                        }
                    }
                    if(is_neg && minRow == undefined || max<Math.abs(matrix[i]
[matrix[i].length-1])){
                        max = Math.abs(matrix[matrix.length-1][i]);
                        minRow = i;
                    }
                }
            }

        var min     = undefined; // minimal division
        var minCell = undefined; // row with minimal division
```

```
            if(minRow == undefined){
                result = Array();
                result['error'] = 'Has no solution by simplex method';
                return result;
            }

            for(j=0;j<matrix[minRow].length-1;j++){
                if(matrix[minRow][j] < 0){
                    if(minCell == undefined || min>Math.abs(matrix[matrix.length-1]
[j]/matrix[minRow][j])){
                        min = Math.abs(matrix[matrix.length-1][j]/matrix[minRow][j]);
                        minCell = j;
                    }
                }
            }
            if(params['show_steps'])
                $("#resultArea").html($("#resultArea").html()+'<p>'+minRow+'
'+minCell+"</p>");

            basis[minRow] = minCell;
            //Next step
            var tmpMatrix = clone(matrix);
            for(i=0;i<matrix.length;i++)
                for(j=0;j<matrix[i].length;j++){
                    if(i != minRow)
                        matrix[i][j] = (tmpMatrix[i][j]*tmpMatrix[minRow][minCell]-
tmpMatrix[minRow][j]*tmpMatrix[i][minCell])/tmpMatrix[minRow][minCell];
                    else
                        matrix[i][j] = tmpMatrix[i][j]/tmpMatrix[minRow][minCell];
                }
```

```
            //Show steps
            if(params['show_steps'] == 1){
                var str = "<table border=1>";
                for(i=0;i<matrix.length;i++){
                    str += "<tr>";
                    str += "<td>"+basis[i]+"</td>";
                    str += "<td>";
                    str += matrix[i].map(function(x){return
Math.round(x*100)/100;}).join('</td><td>');
                    str += "</td></tr>";
                }
                str += "</table>";
                $("#resultArea").html($("#resultArea").html()+str+"<br />");
            }
        }

        result['Game_price'] = 0;
        result['player2_strategies'] = Array(significant_elements);
        for(i=0;i<significant_elements;i++)
            result['player2_strategies'][i] = 0;
        for(i=0;i<matrix.length;i++){
            if(basis[i]<significant_elements){
                result['player2_strategies'][basis[i]] = matrix[i][matrix[i].length-1];
                result['Game_price'] += matrix[i][matrix[i].length-1];
            }
        }

        player1_strategies_len = matrix[0].length-significant_elements-1;
        result['player1_strategies'] = Array(player1_strategies_len);
        for(i=0;i<player1_strategies_len;i++)
            result['player1_strategies'][i] = -matrix[matrix.length-1]
```

```
[significant_elements+i];

            result['Game_price'] = 1/result['Game_price'];

            for(i=0;i<significant_elements;i++)
                result['player2_strategies'][i] = Math.round((result['Game_price'] *
result['player2_strategies'][i] )*100)/100;
            for(i=0;i<player1_strategies_len;i++)
                result['player1_strategies'][i] = Math.round((result['Game_price'] *
result['player1_strategies'][i] )*100)/100;

            result['Game_price'] = Math.round(result['Game_price']*100)/100;

            /*if(maxNegative<0){
                result['Game_price'] -= summand;
            }*/

            return result;
        }
```

Файл server/index.js

```
        var fs = require('fs');
        var vm = require('vm');
        var includeInThisContext = function(path) {
            var code = fs.readFileSync(path);
            vm.runInThisContext(code, path);
        }.bind(this);
        includeInThisContext(__dirname+"/../client/matrixgames/helpers.js");
        includeInThisContext(__dirname+"/../client/matrixgames/symplex.js");
        includeInThisContext(__dirname+"/../client/matrixgames/BraunRobinson.js");
        includeInThisContext(__dirname+"/../client/matrixgames/Nx2xN.js");
```

```
includeInThisContext(__dirname+"/../client/matrixgames/calculate.js");

var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var httpHandle = {};
var socketHandle = {}
httpHandle["/calculate"] = requestHandlers.httpCalculate;
socketHandle["calculate"] = requestHandlers.socketCalculate;
socketHandle["message"] = requestHandlers.socketPostFile;

server.start(router, httpHandle, socketHandle);
```

Файл server/nodejs_paint_canvas_graph.js

```
/* This function needs node-o3-canvas

git://github.com/ajaxorg/node-o3-canvas.git*/

function nodejs_paint_canvas_graph(max, min, maxX, targY, matrix, n2){

    /* Check for 2*n or n*2 */
    if((matrix.length == 2 && matrix[0].length > 1) || (matrix[0].length == 2
&& matrix.length>1)){

        var Canvas = require('canvas')
          , canvas = new Canvas(300,400)
          , ctx = canvas.getContext('2d');

        max    = parseFloat(max);
        min    = parseFloat(min);
```

```
maxX   = parseFloat(maxX);
targY  = parseFloat(targY);

if (canvas.getContext){

    ctx.fillStyle="#000";

    ctx.moveTo(20,20);
    ctx.lineTo(20,390);
    ctx.moveTo(20,20);
    ctx.lineTo(17,27);
    ctx.moveTo(20,20);
    ctx.lineTo(23,27);
    ctx.font = "15px Arial";
    ctx.fillText("u", 0, 30);

    ctx.moveTo(280,20);
    ctx.lineTo(280,390);

    var price = 280/(max-min);
    if(max>0 && min<0){
        ctx.moveTo(0,  380+min*price);
        ctx.lineTo(300,380+min*price);
        ctx.moveTo(300,380+min*price);
        ctx.lineTo(287,380+min*price-3);
        ctx.moveTo(300,380+min*price);
        ctx.lineTo(287,380+min*price+3);
    }

    for(i=0;i<2;i++)
        for(j=0;j<matrix[i].length;j++){
```

```
                    ctx.moveTo(20+i*260-3,380+(-matrix[i][j]+min)*price);
                    ctx.lineTo(20+i*260+3,380+(-matrix[i][j]+min)*price);
                    ctx.fillText(matrix[i][j], 20+i*260-20,380+(-matrix[i][j]
+min)*price);
                    ctx.fillText((i==0?'A':'B')+j, i*260+(i==20?0:25),380+(-matrix[i]
[j]+min)*price);
                }

            if(matrix[0].length == 2 && matrix.length != 2)
                matrix = transpose(matrix);

            for(i=0;i<matrix[0].length;i++){
                ctx.moveTo(20,380+(-matrix[0][i]+min)*price);
                ctx.lineTo(280,380+(-matrix[1][i]+min)*price);
            }

            ctx.fillStyle="#ff0000";

            ctx.moveTo(280*maxX+11, 380+(-targY+min)*price);
            ctx.arc(280*maxX+11, 380+(-targY+min)*price, 3, 0, 2*Math.PI,
false);
            ctx.stroke();
            ctx.fill();
        }

        return  "\n\n"+
            "<img style=\"display:block\" alt=\"Embedded Image\" src=\""+
                canvas.toDataURL()
            +"\" />"
    }
}
```

```
exports.nodejs_paint_canvas_graph = nodejs_paint_canvas_graph;
```

Файл server/requestHandlers.js

```
function httpCalculate(response, request){

    response.write("");
    response.end();
}

function socketCalculate(socket, data){

    matrix = eval(data['matrix']);
    params = JSON.parse(data['params']);
    method = data['method'];
    console.log(data['matrix']);
    var result = calculate(matrix, method, params);

    if(method == 'graph'){
        result['output'] +=
require("./nodejs_paint_canvas_graph").nodejs_paint_canvas_graph(result['max'],
result['min'], result['maxX'], result['targY'], matrix, result['n2']);
    }

    socket.emit('result', JSON.stringify(result) );
}

function socketPostFile(socket, data){
    if(typeof(data) == 'string'){
        data = eval('(' + data + ')');
```

```
            var matrix_str = new Buffer(data['matrix'].substr(37),
'base64').toString('utf8');

            var matrix = matrix_str.split("\n");

            for(i=0;i<matrix.length;i++)
                matrix[i] = matrix[i].split(" ");

            for(i=0;i<matrix.length;i++)
                for(j=0;j<matrix[i].length;j++)
                    matrix[i][j] = parseFloat(matrix[i][j]);
        }else{
            var matrix = eval(data['matrix']);
        }


        params = JSON.parse(data['params']);
        method = data['method'];

        var result = calculate(matrix, method, params);

        if(method == 'graph'){
            result['output'] +=
require("./nodejs_paint_canvas_graph").nodejs_paint_canvas_graph(result['max'],
result['min'], result['maxX'], result['targY'], matrix, result['n2']);
        }

        socket.emit('result', JSON.stringify(result) );
    }

    exports.httpCalculate   = httpCalculate;
```

```
exports.socketCalculate = socketCalculate;
exports.socketPostFile = socketPostFile;
```

Файл server/router.js

```
function httpRoute(handle, pathname, response, request) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response, request);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/html"});
    response.write("404 Not found");
    response.end();
  }
}

function socketRoute(socket, socketHandle) {
    for(var action in socketHandle){
        socket.on(action, function(data){socketHandle[action](socket, data)});
    }
}
exports.httpRoute   = httpRoute;
exports.socketRoute = socketRoute;
```

Файл server/server.js

```
var http = require("http");
var url  = require("url");
function start(router, httpHandle, socketHandle) {
    function onRequest(request, response) {
      var pathname = url.parse(request.url).pathname;
      console.log("Request for " + pathname + " received.");
```

```
      router.httpRoute(httpHandle, pathname, response, request);
    }
    var app = http.createServer(onRequest).listen(8888);
    io = require('socket.io').listen(app);
    io.sockets.on('connection', function (socket) {
        router.socketRoute(socket, socketHandle);
    });
    console.log("Server has started.");
}
exports.start = start;
```