Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до виконаної лабораторної роботи № 6

з дисципліни

“Дослідження операцій”

на тему:

**«Нелінійне програмування: задачі дробово-лінійного програмування»**

Виконав

студент групи *ОІ-11 сп*

*Вальчевський П. В.*

Викладач:

*Сенета М. Я.*

Львів – 2024

## Лабораторна робота № 6

*Тема роботи:*  **Нелінійне програмування: задачі дробово-лінійного програмування.**

*Мета роботи:***ознайомлення з задачами дробово-лінійного програмування, набуття навиків їх аналізу та розв’язку, набуття навиків розв’язку задач дробово-лінійного програмування за допомогою математичних пакетів та розробки оригінальної програми.**

***Завдання***

* + Індивідуальне завдання (номер завдання відповідає двом останнім цифрам залікової книжки студента, крім цифр 00 – які відповідають завданню під номером 100).
  + Розв’язати графічно задану задачу дробово-лінійного програмування засобами Excel.
  + Звести задану задачу до задачі лінійного програмування.
  + Використовуючи один із математичних пакетів ( наприклад, SimplexWin ), розв’язати задачу дробово-лінійного програмування після її зведення до лінійної.
  + Розробити оригінальну програму для розв’язання задачі дробово-лінійного програмування, додавши до програми симплекс-методу процедуру зведення задачі дробово-лінійного програмування до лінійного програмування.

***Порядок виконання роботи***

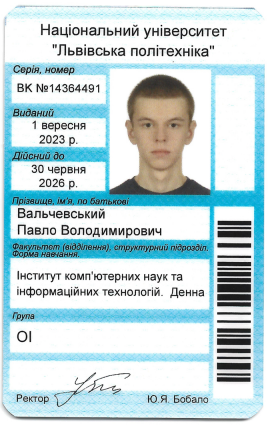
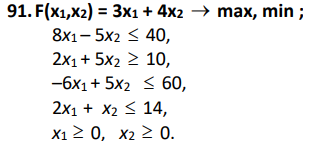
1. **Номер варіанту – 91.**
2. ****

Рис. 1 Фото мого студентського квитка для варіанту ЛР.

1. **Умова завдання:**
2. 

**Розв’язок вручну**

**Зведення задачі дробово-лінійного програмування до задачі лінійного програмування**

**Обрахунок за допомогою симплекс методу зведеної задачі**

Для максимуму

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **C1** | 0 | 3 | 4 | 0 | 0 | 0 | 0 |  |
|  | **Базис** |  |  |  |  |  |  |  |  |
| T1 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
|  | -40 | 8 | -5 | 1 | 0 | 0 | 0 | 0 |
| 0 | 10 | -2 | -5 | 0 | 1 | 0 | 0 | 0 |
| 0 | -60 | -6 | 5 | 0 | 0 | 1 | 0 | 0 |
| 0 | -14 | 2 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
|  | -40 | 18 | 0 | 1 | 0 | 0 | 0 | 5 |
| 0 | 10 | 8 | 0 | 0 | 1 | 0 | 0 | 5 |
| 0 | -60 | -16 | 0 | 0 | 0 | 1 | 0 | -5 |
| 0 | -14 | 0 | 0 | 0 | 0 | 0 | 1 | -1 |
| T2 | 4 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 28.6 | 0 | 1 | 0 | -0.6 | 0 | 8.33 |
| 0 | 0 | 5.3 | 0 | 0 | 1 | 0.16 | 0 | 4.166 |
| 0 | 1 | 0.26 | 0 | 0 | 0 | -0.016 | 0 | 0.083 |
| 0 | 0 | 3.73 | 0 | 0 | 0 | -0.233 | 1 | 0.166 |
|  | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 4 |

Для мінімуму

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **C1** | 0 | -3 | -4 | 0 | 0 | 0 | 0 |  |
|  | **Базис** |  |  |  |  |  |  |  |  |
| T1 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | -40 | 8 | -5 | 1 | 0 | 0 | 0 | 0 |
| 0 | 10 | -2 | -5 | 0 | 1 | 0 | 0 | 0 |
| 0 | -60 | -6 | 5 | 0 | 0 | 1 | 0 | 0 |
| 0 | -14 | 2 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
|  | -40 | 18 | 0 | 1 | 0 | 0 | 0 | 5 |
| 0 | 10 | 8 | 0 | 0 | 1 | 0 | 0 | 5 |
| 0 | -60 | -16 | 0 | 0 | 0 | 1 | 0 | -5 |
| 0 | -14 | 0 | 0 | 0 | 0 | 0 | 1 | -1 |
| T2 | -4 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 28.6 | 0 | 1 | 0 | -0.6 | 0 | 8.33 |
| 0 | 0 | 5.3 | 0 | 0 | 1 | 0.16 | 0 | 4.166 |
| 0 | 1 | 0.26 | 0 | 0 | 0 | -0.016 | 0 | 0.083 |
| 0 | 0 | 3.73 | 0 | 0 | 0 | -0.233 | 1 | 0.166 |
|  | 0 | -5 | 0 | 0 | 0 | 0 | 0 | -4 |
| T3 | -4 | 0 | 0 | 1 | 0 | 0 | 0.125 | -0.535 | 0.910 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1.125 | -7,678 | 7.0535 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0.5 | -1.428 | 3.9285 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | -0.0714 | 0.0714 |
| -3 | 0 | 1 | 0 | 0 | 0 | -0.0625 | 0.2678 | 0.0446 |
|  | 0 | 0 | 0 | 0 | 0 | -0.3125 | 1.339 | -3.776 |
| T4 | -4 | 0 | 0 | 1 | -0.111 | 0 | 0 | 0.317 | 0.1269 |
| 0 | 0 | 0 | 0 | 0.888 | 0 | 1 | -6,825 | 6.2698 |
| 0 | 0 | 0 | 0 | -0.444 | 1 | 0 | 1.9841 | 0.793 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | -0.0714 | 0.0714 |
| -3 | 0 | 1 | 0 | 0.055 | 0 | 0 | -0.1587 | 0.436 |
|  | 0 | 0 | 0 | 0.277 | 0 | 0 | -0.7936 | -1.817 |
| T5 | -4 | 0 | 0 | 1 | -0.04 | -0.16 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | -0.64 | 3.44 | 1 | 0 | 9 |
| 0 | 0 | 0 | 0 | -0.224 | 0.504 | 0 | 1 | 0.4 |
| 0 | 1 | 0 | 0 | -0.016 | 0.036 | 0 | 0 | 0.1 |
| -3 | 0 | 1 | 0 | 0.02 | 0.08 | 0 | 0 | 0.5 |
|  | 0 | 0 | 0 | 0.1 | 0.4 | 0 | 0 | -1.5 |

**Розв’язок графічним методом за допомогою Excel**

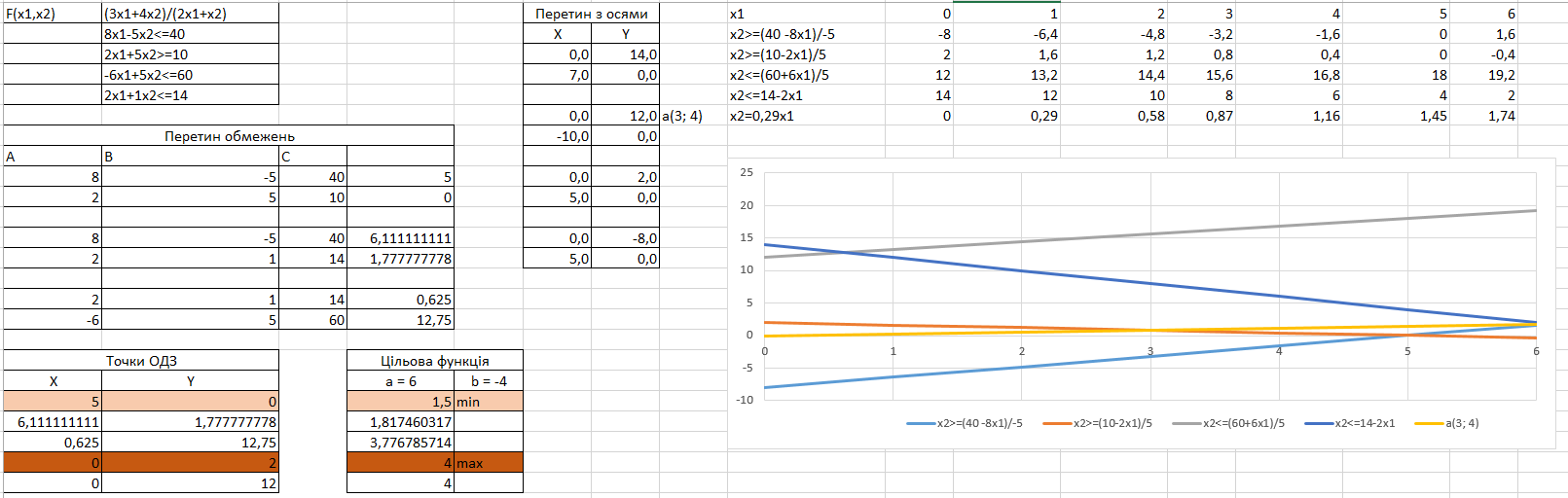
****

Рис. Знімок з Excel.

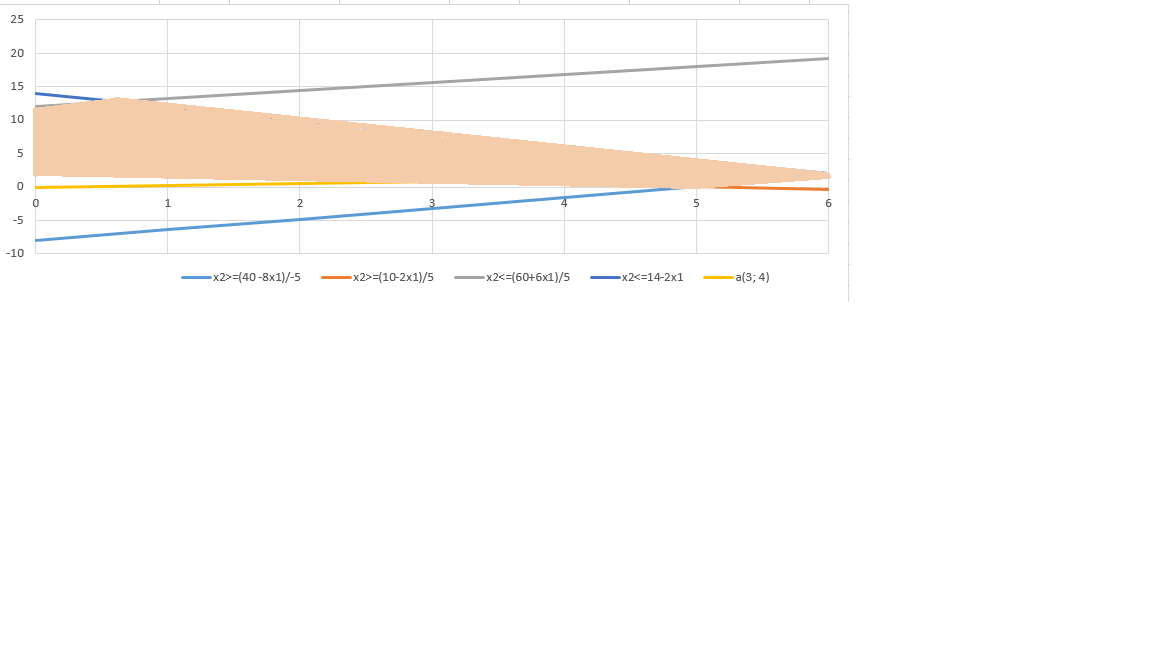
****

Рис. 3 Показ розв'язку з Excel.

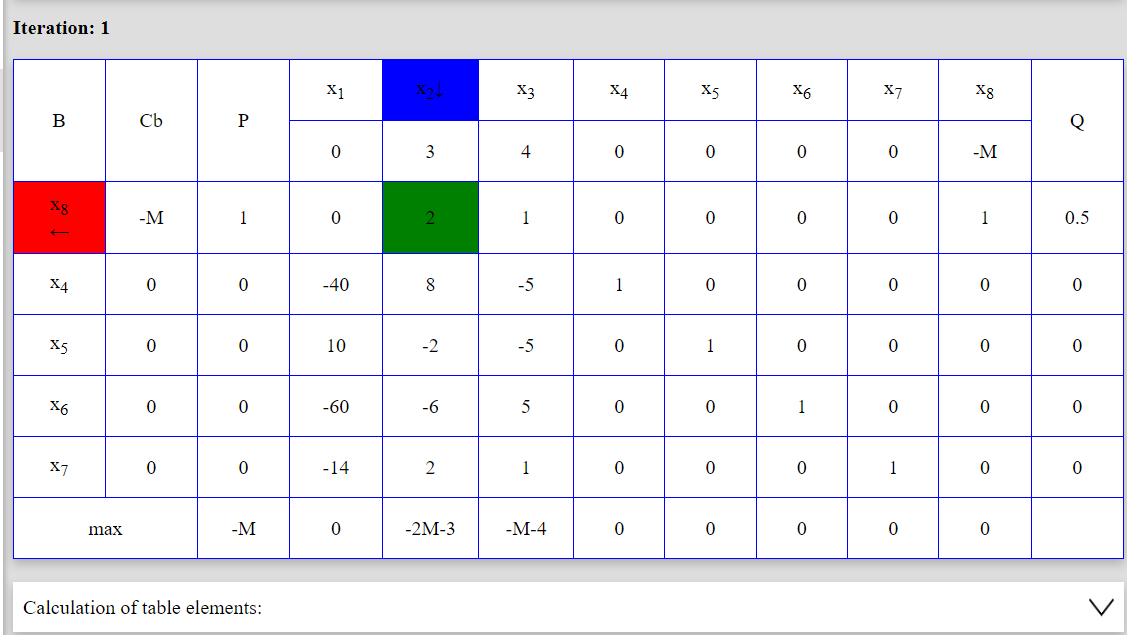
Функція є зростаючою, отже перший дотик при обертанні проти годинникової стрілки дає мінімум, останній – максимум

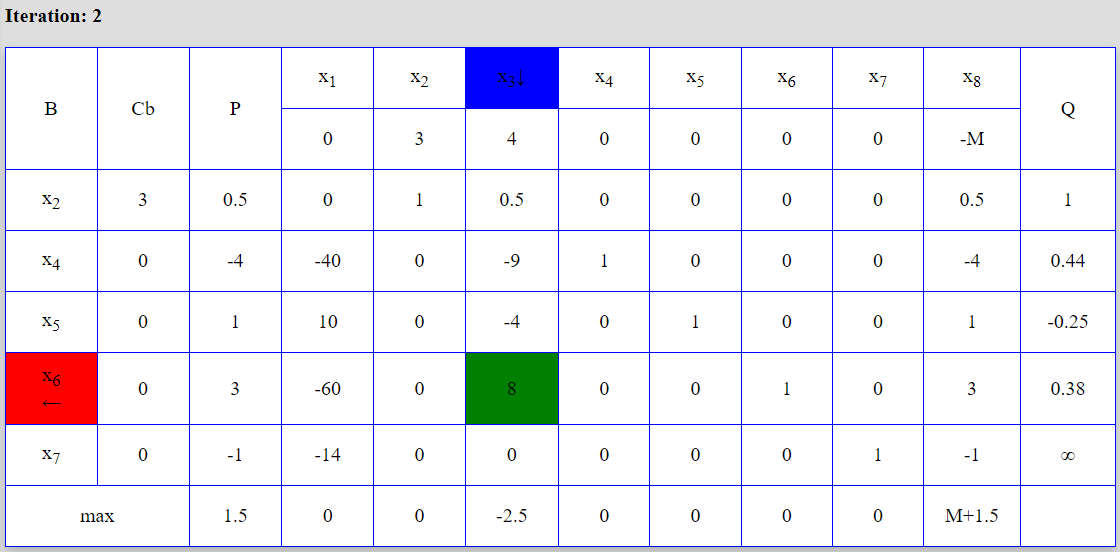
F(A) = 1.5 – min, B(5,0)

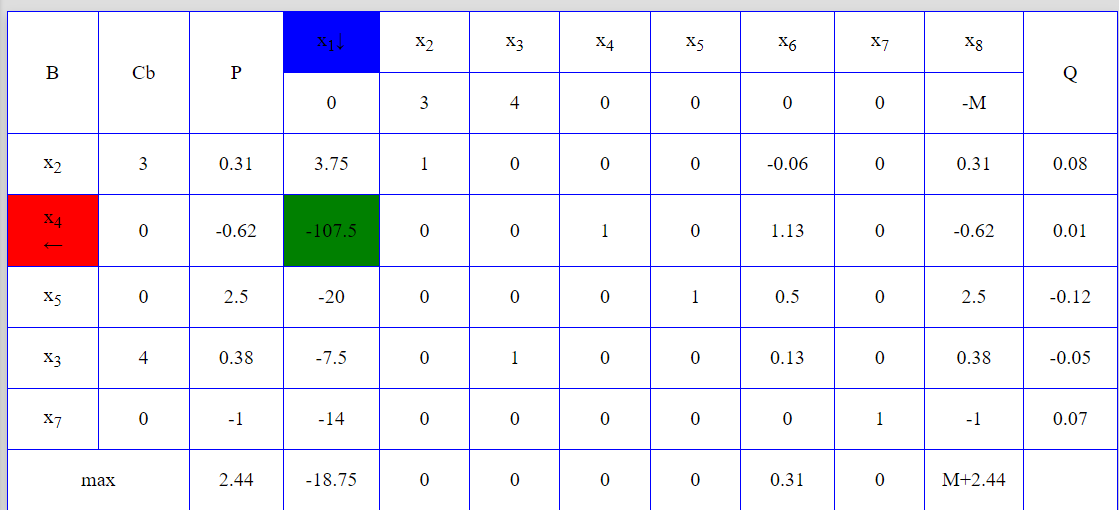
F(D) = 4 – max, C(0, 2)

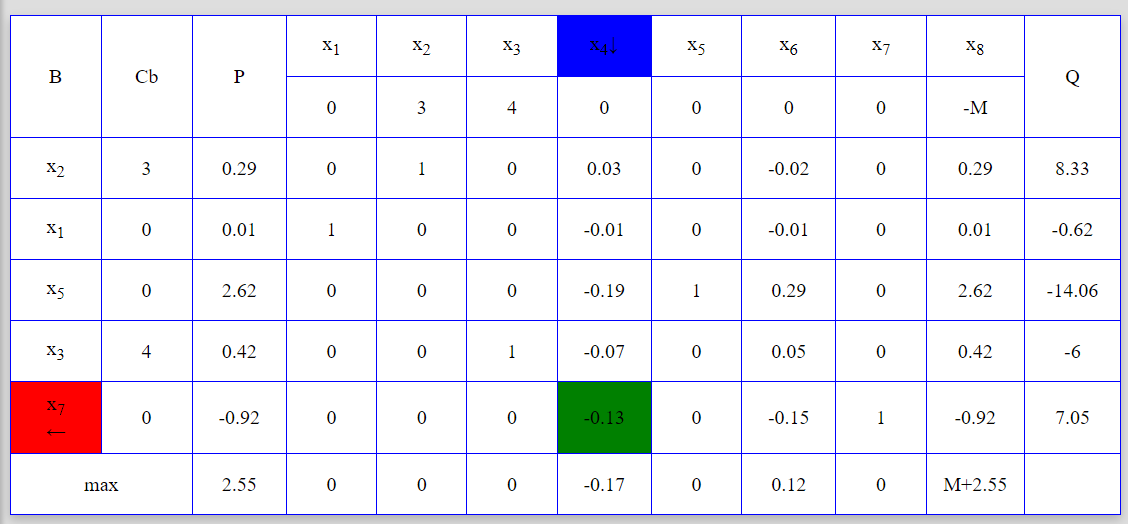
**Розв’язок за допомогою математичного пакету**

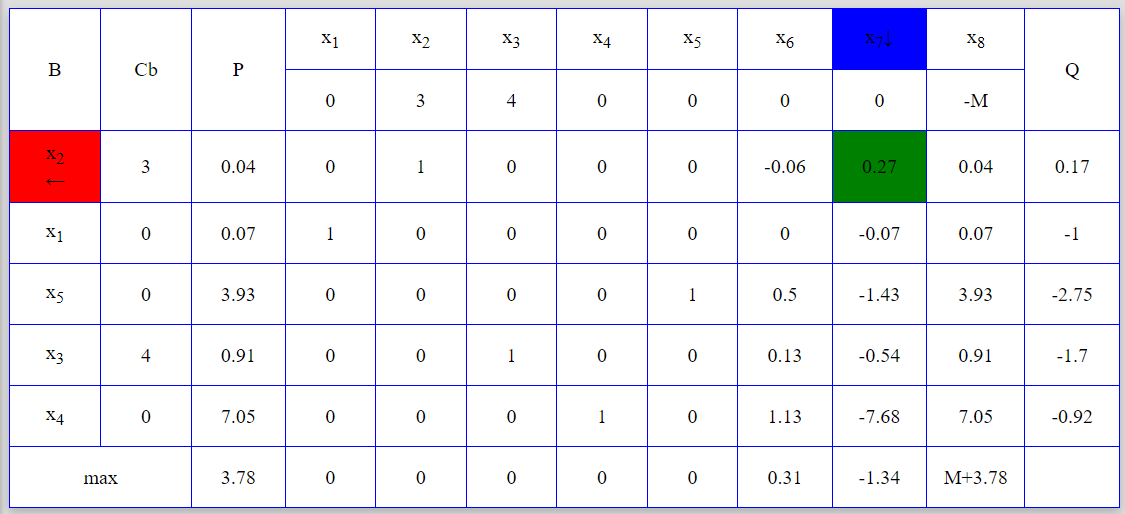
Сайт: https://linprog.com/ru/main-simplex-method

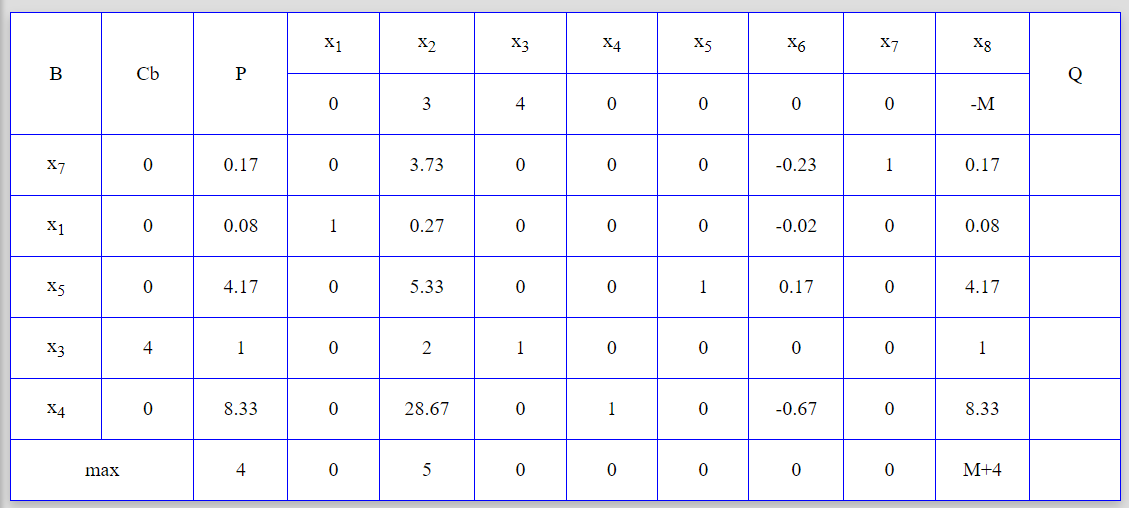












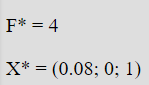
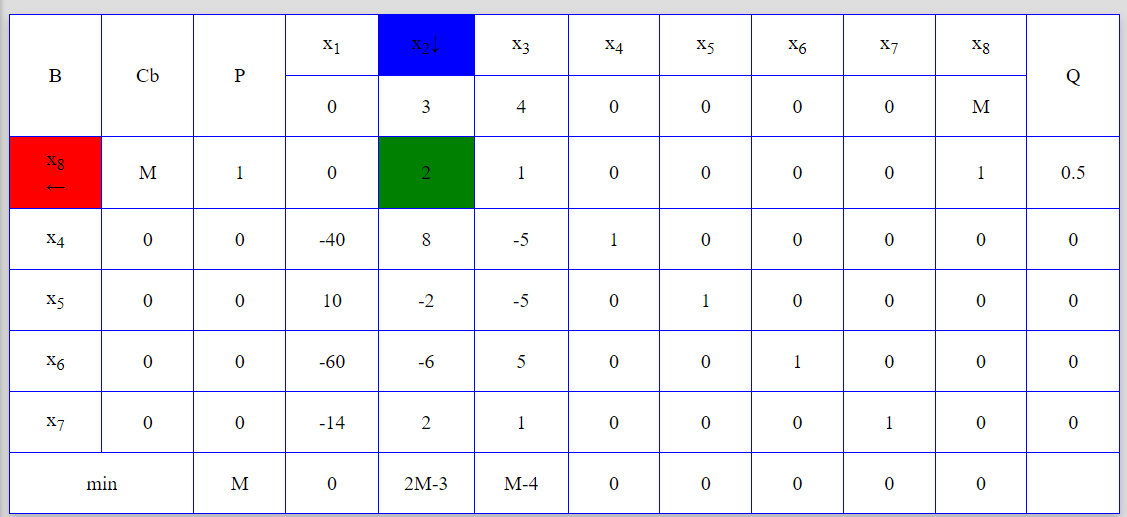
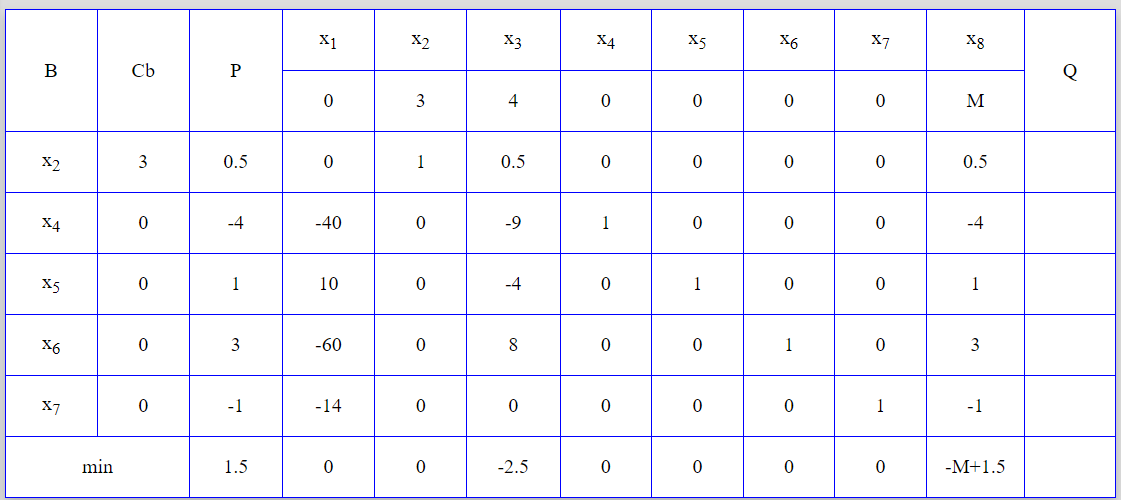


Рис. 4 Результат на максимум через ПЗ.





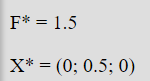


Рис. 5 Результат на мінімум через ПЗ.

**Програмна реалізація**

Function.java

package org.example;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
*/\*  
У цьому класі визначається об'єкт, що представляє функцію з коефіцієнтами, значенням та знаком. Клас містить конструктори для ініціалізації об'єктів, методи для додавання, ділення та множення функцій на число, а також методи для перетворення коефіцієнтів та значення у рядок. Клас також містить вкладений перерахування для представлення можливих знаків функцій.  
  
Основні елементи класу:  
  
coefficients: Список коефіцієнтів функції.  
value: Значення функції.  
sign: Знак функції (GREATER, EQUAL, LESS).  
Методи виконують операції з об'єктами типу Function:  
  
add: Додає іншу функцію, помножену на певне число, до поточної функції.  
divide: Ділить кожен коефіцієнт та значення функції на задане число.  
multiply: Перемножує кожен коефіцієнт та значення функції на задане число.  
Також є методи для перетворення коефіцієнтів та значення у рядок.  
  
Клас містить вкладений перерахування для представлення можливих знаків функцій (більше або дорівнює, дорівнює, менше або дорівнює).  
\*/*public class Function {  
 List<Double> coefficients;  
 double value;  
 Sign sign;  
  
 public Function(Double[] coefficients, double value, Sign sign) {  
 this.coefficients = new ArrayList<>(Arrays.*asList*(coefficients));  
 this.value = value;  
 this.sign = sign;  
 }  
  
 public Function(Double[] coefficients) {  
 this.coefficients = new ArrayList<>(Arrays.*asList*(coefficients));  
 sign = Sign.*EQUAL*;  
 }  
  
 public Function(Function function) {  
 coefficients = new ArrayList<>(function.coefficients);  
 value = function.value;  
 sign = function.sign;  
 }  
  
 public void add(Function function, double x) {  
 for (int i = 0; i < coefficients.size(); i++) {  
 coefficients.set(i, coefficients.get(i) + function.coefficients.get(i) \* x);  
 }  
 value += function.value \* x;  
 }  
  
 public void divide(double x) {  
 coefficients.replaceAll(aDouble -> aDouble / x);  
 value /= x;  
 }  
  
 public void multiply(double x) {  
 coefficients.replaceAll(aDouble -> aDouble \* x);  
 value \*= x;  
 if (x < 0) {  
 sign = sign.opposite();  
 }  
 }  
  
 public String coefToString() {  
 StringBuilder builder = new StringBuilder("|");  
 coefficients.forEach((a) -> builder.append("%10.5f|".formatted(a)));  
 return builder.toString();  
 }  
  
 @Override  
 public String toString() {  
 return coefToString() + "%10.5f|".formatted(value);  
 }  
  
 public enum Sign {  
 *GREATER*(">="),  
 *EQUAL*("="),  
 *LESS*("<=");  
  
 private final String symbol;  
  
 Sign(String symbol) {  
 this.symbol = symbol;  
 }  
  
 Sign opposite() {  
 return switch (this) {  
 case *GREATER* -> *LESS*;  
 case *EQUAL* -> *EQUAL*;  
 case *LESS* -> *GREATER*;  
 };  
 }  
  
 @Override  
 public String toString() {  
 return symbol;  
 }  
 }  
}

FunctionSystem.java

package org.example;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.StringJoiner;  
  
*/\*  
У цьому класі реалізовано систему лінійних функцій, яка використовується для оптимізації методом двоїстого симплекс-методу. Основні функції та методи класу включають:  
  
Ініціалізація системи функцій:  
  
Конструктори для створення нових систем функцій або копіювання існуючих.  
Метод getDual, який повертає дуальну систему функцій.  
Метод getOptimal, який знаходить оптимальне значення функції в системі.  
Методи getOptimalDual та getOptimalInt, які знаходять оптимальне значення для двоїстого симплекс-методу.  
Додавання функцій до системи:  
  
Метод addFunction, який додає нову функцію до системи.  
Метод addFunctionComplete, який додає повну функцію до системи.  
Виконання оптимізації:  
  
Методи findOptimalDualSimplex та getOptimal, які виконують оптимізацію за допомогою симплекс-методу.  
Методи setBasis, minRow, toOne, які допомагають в процесі оптимізації.  
Додаткові допоміжні методи:  
  
Методи для роздруківки результатів (print, printHeader, printFunctions, printZCDiff, printResult).  
Методи для обчислення значень та множення векторів (getObjectiveValue, getVectorMultiplication).  
Цей клас дозволяє моделювати системи лінійних функцій та знаходити їхні оптимальні значення за допомогою симплекс-методу.  
 \*/*public class FunctionSystem {  
 protected String delimiter;  
 protected final Function objective;  
 protected final List<Function> functions = new ArrayList<>();  
 protected int prevCoefNum;  
 protected int coefNum;  
 protected final int orgSize;  
 protected boolean inverse;  
 protected List<Double> res;  
 protected final List<Integer> basis;  
 protected final List<Double> basisCoef;  
 protected final List<Double> zcDiff;  
  
 public FunctionSystem(Function objective) {  
 this.objective = objective;  
 coefNum = orgSize = objective.coefficients.size();  
 prevCoefNum = -1;  
 res = new ArrayList<>();  
 basis = new ArrayList<>();  
 basisCoef = new ArrayList<>();  
 zcDiff = new ArrayList<>();  
 inverse = false;  
 }  
  
 public FunctionSystem(FunctionSystem system) {  
 objective = new Function(system.objective);  
 for (Function f : system.functions) {  
 functions.add(new Function(f));  
 }  
 prevCoefNum = system.prevCoefNum;  
 coefNum = system.coefNum;  
 orgSize = system.orgSize;  
 inverse = system.inverse;  
 delimiter = system.delimiter;  
 res = new ArrayList<>(system.res);  
 basis = new ArrayList<>(system.basis);  
 basisCoef = new ArrayList<>(system.basisCoef);  
 zcDiff = new ArrayList<>(system.zcDiff);  
 }  
  
 public FunctionSystem getDual(boolean max) {  
 FunctionSystem system = new FunctionSystem(this);  
 Double[][] dualFunctions = new Double[system.orgSize][system.functions.size()];  
 Double[] dualValues = new Double[system.orgSize];  
 Double[] dualObjective = new Double[system.functions.size()];  
  
 if (max) {  
 system.objective.multiply(-1);  
 }  
  
 for (int i = 0; i < system.functions.size(); i++) {  
 if (system.functions.get(i).sign == Function.Sign.*LESS*) {  
 system.functions.get(i).multiply(-1);  
 }  
  
 for (int j = 0; j < system.orgSize; j++) {  
 dualFunctions[j][i] = system.functions.get(i).coefficients.get(j);  
 }  
 dualObjective[i] = system.functions.get(i).value;  
 }  
 for (int i = 0; i < system.orgSize; i++) {  
 dualValues[i] = system.objective.coefficients.get(i);  
 }  
  
 system = new FunctionSystem(new Function(dualObjective));  
 for (int i = 0; i < dualFunctions.length; i++) {  
 system.addFunction(new Function(dualFunctions[i], dualValues[i], Function.Sign.*LESS*));  
 }  
 system.inverse = false;  
 return system;  
 }  
  
 public List<Double> getOptimalDual() {  
 return getOptimal(true);  
 }  
  
 public List<Double> getOptimalInt(boolean max) {  
 prevCoefNum = -1;  
 getOptimal(max);  
 if (res.isEmpty()) {  
 return null;  
 }  
  
 List<Double> valuesFractional = new ArrayList<>();  
 for (Function function : functions) {  
 valuesFractional.add(getFractional(function.value));  
 }  
  
 int indexMax;  
 try {  
 indexMax = optimalIndFractional(valuesFractional, true);  
 } catch (Exception e) {  
 res.clear();  
 printResult();  
 return null;  
 }  
  
 if (indexMax != -1) {  
 System.*out*.println("Result contains non-integer values. We need to use Gomori's method");  
 }  
 while (indexMax != -1) {  
 int ind = indexMax;  
 Double[] newRow = new Double[coefNum];  
 Arrays.*setAll*(newRow, (i) -> {  
 double fractional = functions.get(ind).coefficients.get(i);  
 fractional = getFractional(fractional);  
 if (fractional < 0.00001) {  
 return 0.0;  
 } else {  
 return -fractional;  
 }  
 });  
  
 addFunction(new Function(newRow, -valuesFractional.get(indexMax), Function.Sign.*LESS*));  
 basis.add(coefNum - 1);  
 basisCoef.add(0.0);  
 if (findOptimalDualSimplex() == null) {  
 res.clear();  
 printResult();  
 return null;  
 }  
  
 for (int i = 0; i < functions.size(); i++) {  
 double fractional = getFractional(functions.get(i).value);  
 fractional = fractional < 0.00001 || fractional > 0.99999 ? 0 : fractional;  
 if (valuesFractional.size() < i + 1) {  
 valuesFractional.add(fractional);  
 } else {  
 valuesFractional.set(i, fractional);  
 }  
 }  
  
 try {  
 indexMax = optimalIndFractional(valuesFractional, true);  
 } catch (Exception e) {  
 res.clear();  
 printResult();  
 return null;  
 }  
 }  
  
 printResult();  
 return res;  
 }  
  
 private double getFractional(double x) {  
 return x - Math.*floor*(x);  
 }  
  
 private int optimalIndFractional(List<Double> list, boolean max)  
 throws Exception {  
 int ind = -1;  
 boolean found = false;  
 for (int i = 0; i < list.size(); i++) {  
 if (list.get(i) != 0 && (ind == -1 ||  
 max && list.get(i) > list.get(ind) ||  
 !max && list.get(i) < list.get(ind))) {  
 found = true;  
 boolean hasFractional = false;  
 for (int j = 0; j < functions.get(i).coefficients.size(); j++) {  
 double fractional = getFractional(functions.get(i).coefficients.get(j));  
 hasFractional = hasFractional || fractional > 0.00001 && fractional < 0.99999;  
 }  
 if (hasFractional) {  
 ind = i;  
 }  
 }  
 }  
  
 if (found && ind == -1) {  
 throw new Exception();  
 }  
 return ind;  
 }  
  
 public List<Double> findOptimalDualSimplex() {  
 setZcDiff();  
 print();  
  
 int minRow = minRow();  
 while (minRow != -1) {  
 int minCol = -1;  
 double minValue = 0;  
 for (int i = 0; i < coefNum; i++) {  
 if (functions.get(minRow).coefficients.get(i) < -0.00001) {  
 double value = -zcDiff.get(i) /  
 functions.get(minRow).coefficients.get(i);  
 if (functions.get(minRow).coefficients.get(i) < -0.00001 &&  
 (minCol == -1 || value < minValue)) {  
 minCol = i;  
 minValue = value;  
 }  
 }  
 }  
  
 if (minCol == -1) {  
 res.clear();  
 return null;  
 }  
 toOne(minRow, minCol);  
 basis.set(minRow, minCol);  
 basisCoef.set(minRow, objective.coefficients.get(minCol));  
 setZcDiff();  
  
 print();  
 minRow = minRow();  
 }  
  
 return setRes();  
 }  
  
 private int minRow() {  
 int minRow = -1;  
 for (int i = 0; i < functions.size(); i++) {  
 if (functions.get(i).value < -0.00001 && (minRow == -1 ||  
 functions.get(i).value < functions.get(minRow).value)) {  
 minRow = i;  
 }  
 }  
 return minRow;  
 }  
  
 public List<Double> getOptimal(boolean max) {  
 getOptimal(max, false);  
 return res;  
 }  
  
 public List<Double> getOptimal(boolean max, boolean additionalVariable) {  
 if (max == inverse) {  
 objective.multiply(-1);  
 inverse = !inverse;  
 }  
 setDelimiter();  
  
 print();  
 if (!setBasis(additionalVariable)) {  
 print();  
 res.clear();  
 printResult();  
 return null;  
 }  
  
 for (Integer ind : basis) {  
 basisCoef.add(objective.coefficients.get(ind));  
 }  
 if (additionalVariable) {  
 print();  
 }  
  
 do {  
 setZcDiff();  
 print();  
  
 int maxNotOpt = -1;  
 for (int i = 0; i < zcDiff.size(); i++) {  
 if (zcDiff.get(i) < 0) {  
 if (maxNotOpt == -1 ||  
 Math.*abs*(zcDiff.get(i)) > Math.*abs*(zcDiff.get(maxNotOpt))) {  
 maxNotOpt = i;  
 }  
 }  
 }  
  
 if (maxNotOpt == -1) {  
 setRes();  
 printResult();  
 return res;  
 }  
  
 int minPropInd = -1;  
 double minProportion = 0;  
 for (int i = 0; i < functions.size(); i++) {  
 double coefficient = functions.get(i).coefficients.get(maxNotOpt);  
 if (coefficient > 0) {  
 double proportion = functions.get(i).value / coefficient;  
 if (minPropInd == -1 || minProportion > proportion) {  
 minPropInd = i;  
 minProportion = proportion;  
 }  
 }  
 }  
  
 if (minPropInd != -1) {  
 toOne(minPropInd, maxNotOpt);  
 basis.set(minPropInd, maxNotOpt);  
 basisCoef.set(minPropInd,  
 objective.coefficients.get(basis.get(minPropInd)));  
 } else {  
 res.clear();  
 printResult();  
 return null;  
 }  
 } while (true);  
 }  
  
 private boolean setBasis(boolean additionalVariable) {  
 basis.clear();  
 basisCoef.clear();  
 for (Function f : functions) {  
 if (f.value < -0.001) {  
 f.multiply(-1);  
 }  
 }  
  
 for (int i = 0; i < functions.size(); i++) {  
 for (int j = 0; j < coefNum; j++) {  
 if (isBasis(i, j)) {  
 basis.add(j);  
 break;  
 }  
 }  
 if (basis.size() == i + 1) {  
 continue;  
 }  
  
 if (additionalVariable) {  
 functions.get(i).coefficients.add(1.0);  
 for (int j = 0; j < functions.size(); j++) {  
 if (j != i) {  
 functions.get(j).coefficients.add(0.0);  
 }  
 }  
 objective.coefficients.add(-100.0);  
 basis.add(++coefNum - 1);  
 } else {  
 int min = -1;  
 for (int j = 0; j < coefNum; j++) {  
 double multiply = functions.get(i).value == 0 ? functions.get(i).coefficients.get(j)  
 : functions.get(i).coefficients.get(j) \* functions.get(i).value;  
 if (Math.*abs*(functions.get(i).coefficients.get(j)) > 0.0001 &&  
 multiply > 0) {  
 if (min == -1 || Math.*abs*(functions.get(i).coefficients.get(j)) < Math.*abs*(functions.get(i).coefficients.get(min))) {  
 min = j;  
 }  
 }  
 }  
 if (min != -1) {  
 toOne(i, min);  
 }  
 basis.add(min);  
 }  
 }  
  
 print();  
 boolean loop;  
 int num = 0;  
 do {  
 loop = false;  
 for (int i = 0; i < functions.size(); i++) {  
 if (functions.get(i).value < -0.0001) {  
 int min = -1;  
 for (int j = 0; j < coefNum; j++) {  
 if (functions.get(i).coefficients.get(j) < -0.0001) {  
 if (min == -1 ||  
 functions.get(i).coefficients.get(j) < functions.get(i).coefficients.get(min)) {  
 min = j;  
 }  
 }  
 }  
 if (min != -1) {  
 toOne(i, min);  
 basis.set(i, min);  
 }  
 loop = true;  
 }  
 }  
 num++;  
 if (num > 100) {  
 return false;  
 }  
 } while (loop);  
  
 for (int b : basis) {  
 if (b == -1) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 private boolean isBasis(int x, int y) {  
 if (functions.get(x).coefficients.get(y) != 1) {  
 return false;  
 }  
 for (int i = 0; i < functions.size(); i++) {  
 if (i != x && Math.*abs*(functions.get(i).coefficients.get(y)) > 0.00001) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 private void setZcDiff() {  
 for (int i = 0; i < coefNum; i++) {  
 if (zcDiff.size() < (i + 1)) {  
 zcDiff.add(getVectorMultiplication(basisCoef, i));  
 } else {  
 zcDiff.set(i, getVectorMultiplication(basisCoef, i));  
 }  
 }  
 }  
  
 protected List<Double> setRes() {  
 res = new ArrayList<>(coefNum);  
 for (int i = 0; i < coefNum; i++) {  
 res.add(0.0);  
 }  
 for (int i = 0; i < basis.size(); i++) {  
 res.set(basis.get(i), functions.get(i).value);  
 }  
 return res;  
 }  
  
 public void setDelimiter() {  
 delimiter = "+" + ("-".repeat(10) + "+").repeat(coefNum + 2);  
 }  
  
 public double getObjectiveValue(List<Double> x) {  
 double res = 0;  
 for (int i = 0; i < x.size(); i++) {  
 res += objective.coefficients.get(i) \* x.get(i);  
 }  
 return res;  
 }  
  
 public void addFunction(Function function) {  
 functions.add(function);  
 for (int i = function.coefficients.size(); i < coefNum; i++) {  
 function.coefficients.add(0.0);  
 }  
 if (function.sign == Function.Sign.*GREATER*) {  
 function.coefficients.add(-1.0);  
 } else if (function.sign == Function.Sign.*LESS*) {  
 function.coefficients.add(1.0);  
 }  
 if (function.sign != Function.Sign.*EQUAL*) {  
 for (Function f : functions) {  
 if (!function.equals(f)) {  
 f.coefficients.add(0.0);  
 }  
 }  
 objective.coefficients.add(0.0);  
 coefNum++;  
 }  
 function.sign = Function.Sign.*EQUAL*;  
 }  
  
 public void addFunctionComplete(Function function) {  
 functions.add(function);  
 }  
  
 public void print() {  
 setDelimiter();  
 if (prevCoefNum != coefNum) {  
 printHeader();  
 prevCoefNum = coefNum;  
 }  
 printFunctions();  
 if (!zcDiff.isEmpty()) {  
 printZCDiff();  
 }  
 }  
  
 public void printHeader() {  
 System.*out*.println(delimiter);  
 System.*out*.println("|%10s".formatted("") +  
 objective.coefToString() + "%10s|".formatted("b"));  
 System.*out*.println(delimiter);  
 }  
  
 public void printFunctions() {  
 for (int i = 0; i < functions.size(); i++) {  
 System.*out*.printf("|%6.1f p%-2d%s%n",  
 basisCoef.isEmpty() ? 0 : basisCoef.get(i),  
 basis.isEmpty() ? 0 : basis.get(i) + 1,  
 functions.get(i).toString());  
 }  
 System.*out*.println(delimiter);  
 }  
  
 private void printZCDiff() {  
 StringBuilder builder = new StringBuilder("|%10s|".formatted(""));  
 for (double v : zcDiff) {  
 builder.append("%10.5f|".formatted(v));  
 }  
 builder.append("%10.5f|".formatted(getVectorMultiplication(basisCoef, coefNum)));  
  
 System.*out*.println(builder);  
 System.*out*.println(delimiter);  
 }  
  
 protected void printResult() {  
 if (res.isEmpty()) {  
 System.*out*.println("There is no " + (!inverse ? "max" : "min") + " value");  
 } else {  
 StringJoiner joiner = new StringJoiner(", ", "[", "]");  
 for (double x : res) {  
 joiner.add("%.3f".formatted(x));  
 }  
 System.*out*.printf((!inverse ? "Max" : "Min") + " point - %s, F" +  
 (!inverse ? "max" : "min") + " = %.3f%n", joiner,  
 inverse ? -getObjectiveValue(res) : getObjectiveValue(res));  
 }  
 System.*out*.println();  
 }  
  
 public double getVectorMultiplication(List<Double> vector, int y) {  
 if (vector.size() != functions.size()) {  
 return Double.*MIN\_VALUE*;  
 }  
 double res;  
 if (y < coefNum) {  
 res = -objective.coefficients.get(y);  
 for (int i = 0; i < vector.size(); i++) {  
 res += functions.get(i).coefficients.get(y) \* vector.get(i);  
 }  
 } else {  
 res = 0;  
 for (int i = 0; i < vector.size(); i++) {  
 res += functions.get(i).value \* vector.get(i);  
 }  
 }  
 return res;  
 }  
  
 public void toOne(int x, int y) {  
 functions.get(x).divide(functions.get(x).coefficients.get(y));  
 for (int i = 0; i < functions.size(); i++) {  
 if (i != x) {  
 functions.get(i).add(functions.get(x), -functions.get(i).coefficients.get(y));  
 }  
 }  
 }  
  
 protected void buildFunction(Function function, StringBuilder builder) {  
 builder.append(function.coefficients.get(0)).append("x1");  
 for (int i = 1; i < function.coefficients.size(); i++) {  
 double x = function.coefficients.get(i);  
 builder.append(x < 0 ? " - " : " + ")  
 .append(Math.*abs*(x)).append('x').append(i + 1);  
 }  
 if (function != objective) {  
 builder.append(" ").append(function.sign)  
 .append(" ").append(function.value);  
 }  
 builder.append("\n");  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder builder = new StringBuilder("F = ");  
 buildFunction(objective, builder);  
 functions.forEach((f) -> buildFunction(f, builder));  
 return builder.toString();  
 }  
}

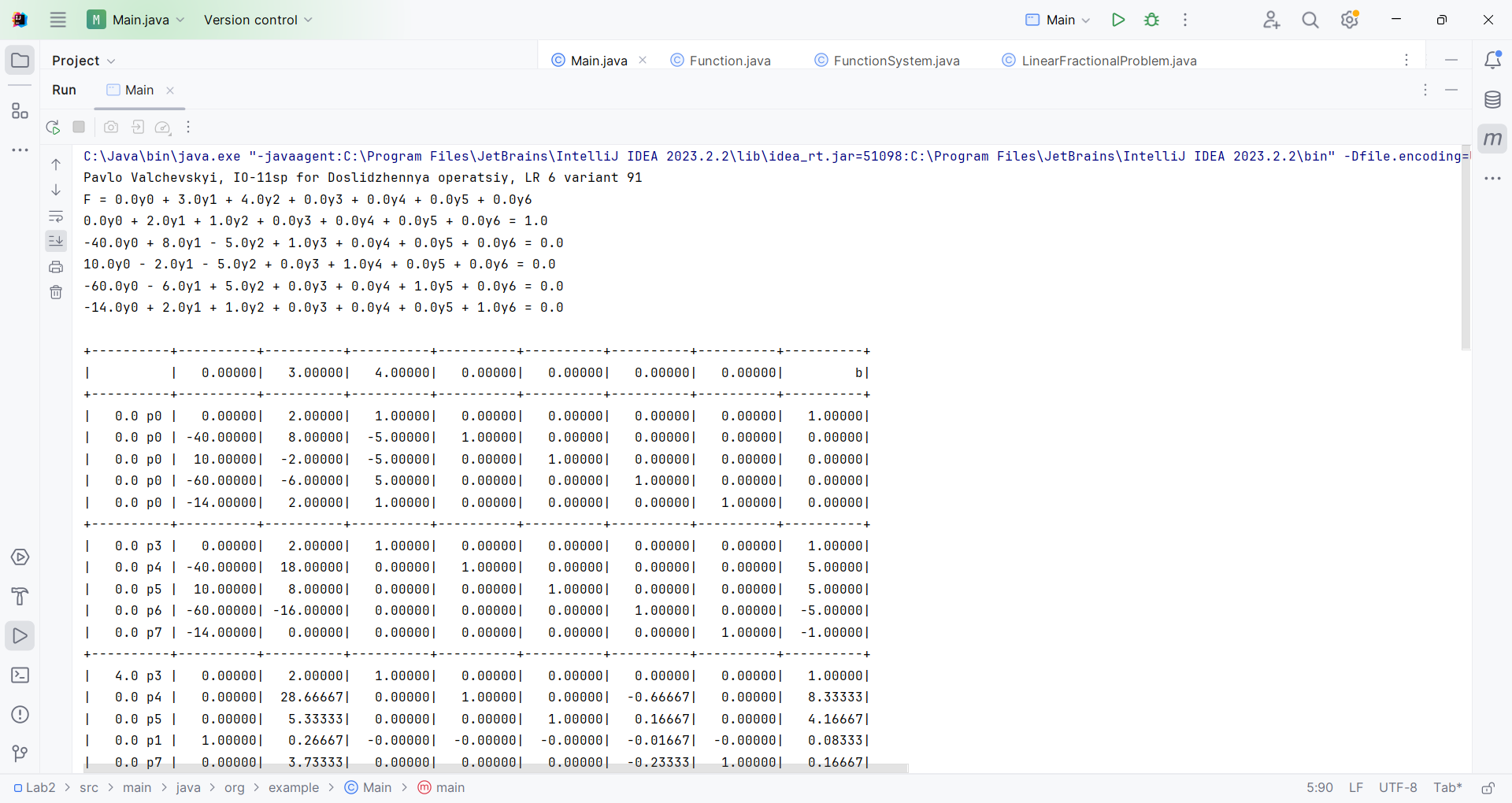
LinearFractionalProblem.java

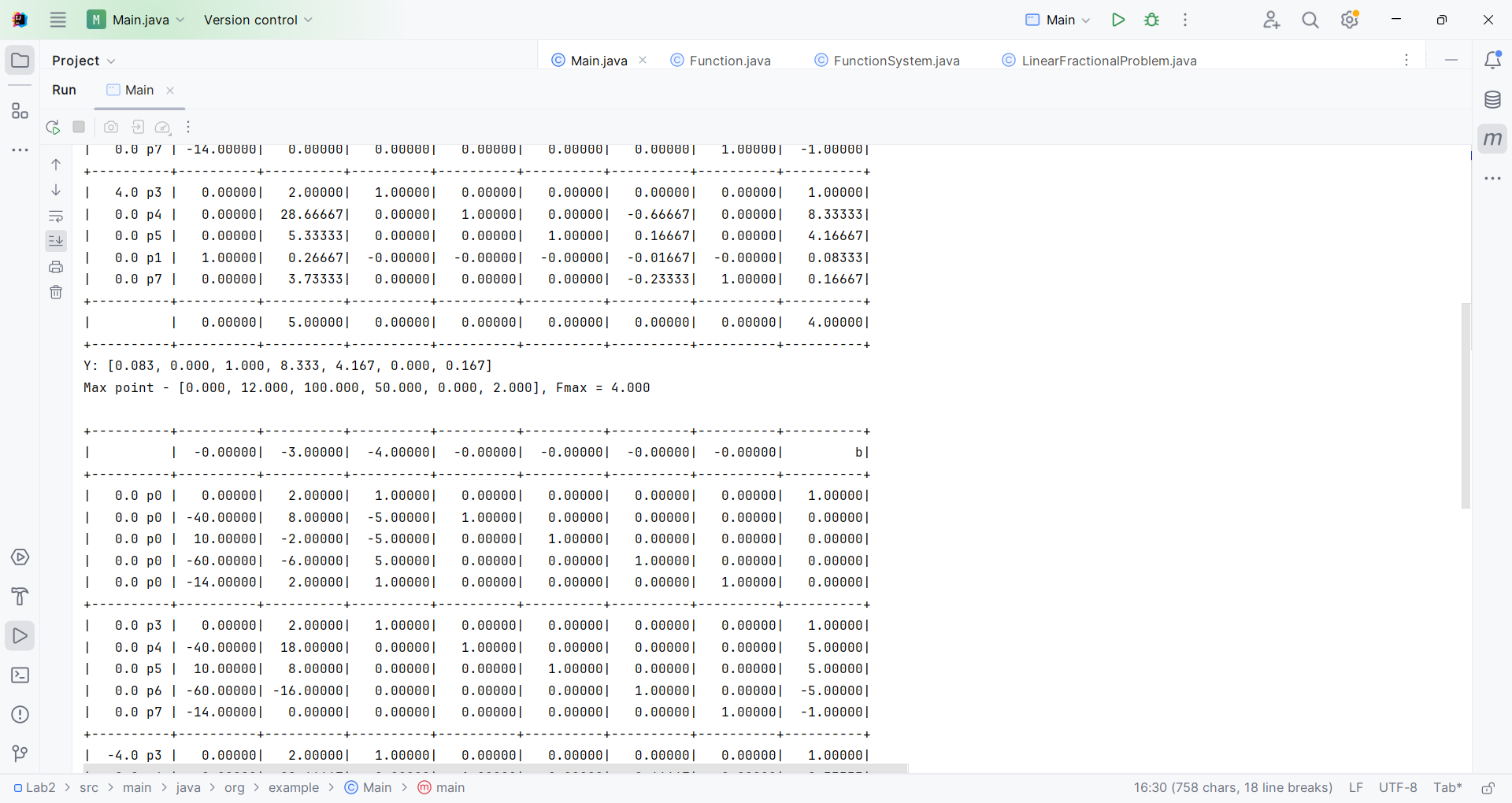
package org.example;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.StringJoiner;  
  
*/\*  
  
У цьому класі реалізована система лінійних дробових задач (Linear Fractional Problems), яка є підкласом класу FunctionSystem. Основні аспекти цього класу включають:  
  
Ініціалізація:  
  
Конструктор, який приймає чисельник та знаменник дробової функції.  
У конструкторі створюються об'єкти функцій з чисельника та знаменника, а також додаткові коефіцієнти для функцій, що додаються.  
Додавання функцій:  
  
Перевизначений метод addFunction, який додає нову функцію до системи.  
Додавання додаткових коефіцієнтів до функцій, які враховують чисельник та знаменник.  
Обчислення результатів:  
  
Перевизначений метод setRes, який обчислює результати для дробової задачі.  
Перевизначений метод printResult, який виводить результати у вигляді дробових значень.  
Цей клас розширює функціональність системи функцій для обробки дробових задач, де функції виражені як дроби, а не просто лінійні функції.  
\*/*public class LinearFractionalProblem extends FunctionSystem {  
 private List<Double> y;  
  
 public LinearFractionalProblem(Double[] numerator, Double[] denominator) {  
 super(new Function(numerator));  
 objective.coefficients.add(0, 0.0);  
  
 functions.add(new Function(denominator, 1, Function.Sign.*EQUAL*));  
 functions.get(0).coefficients.add(0, 0.0);  
 coefNum = denominator.length + 1;  
 }  
  
 public LinearFractionalProblem(LinearFractionalProblem system) {  
 super(system);  
 this.y = system.y;  
 }  
  
 @Override  
 public void addFunction(Function function) {  
 Double[] coefs = new Double[function.coefficients.size() + 1];  
 coefs[0] = function.sign == Function.Sign.*LESS* ? -function.value : function.value;  
 for (int i = 0; i < function.coefficients.size(); i++) {  
 coefs[i + 1] = function.sign == Function.Sign.*LESS* ?  
 function.coefficients.get(i) : -function.coefficients.get(i);  
 }  
 Function func = new Function(coefs, 0, Function.Sign.*LESS*);  
 super.addFunction(func);  
 }  
  
 @Override  
 protected List<Double> setRes() {  
 super.setRes();  
 y = res;  
 res = new ArrayList<>(coefNum - 1);  
 for (int i = 1; i < coefNum; i++) {  
 res.add(y.get(i) / y.get(0));  
 }  
 return res;  
 }  
  
 @Override  
 protected void printResult() {  
 if (res.isEmpty()) {  
 System.*out*.println("There is no " + (!inverse ? "max" : "min") + " value");  
 } else {  
 StringJoiner joiner = new StringJoiner(", ", "[", "]");  
 for (double x : y) {  
 joiner.add("%.3f".formatted(x));  
 }  
 System.*out*.println("Y: " + joiner);  
  
 joiner = new StringJoiner(", ", "[", "]");  
 for (double x : res) {  
 joiner.add("%.3f".formatted(x));  
 }  
 System.*out*.printf((!inverse ? "Max" : "Min") + " point - %s, F" +  
 (!inverse ? "max" : "min") + " = %.3f%n", joiner,  
 inverse ? -getObjectiveValue(y) : getObjectiveValue(y));  
 }  
 System.*out*.println();  
 }  
  
 @Override  
 protected void buildFunction(Function function, StringBuilder builder) {  
 builder.append(function.coefficients.get(0)).append("y0");  
 for (int i = 1; i < function.coefficients.size(); i++) {  
 double x = function.coefficients.get(i);  
 builder.append(x < 0 ? " - " : " + ")  
 .append(Math.*abs*(x)).append('y').append(i);  
 }  
 if (function != objective) {  
 builder.append(" ").append(function.sign)  
 .append(" ").append(function.value);  
 }  
 builder.append("\n");  
 }  
}

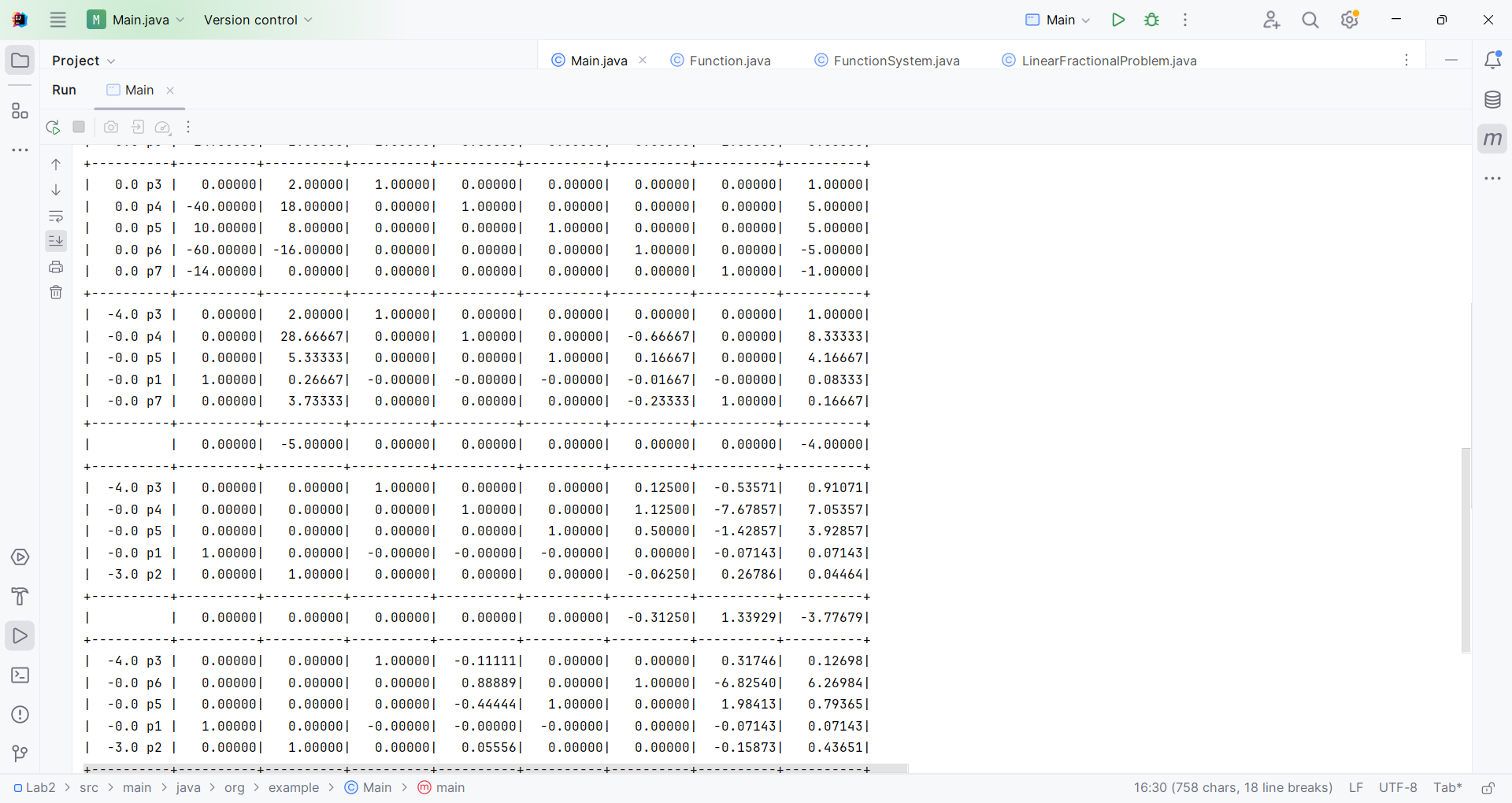
Main.java

package org.example;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Pavlo Valchevskyi, IO-11sp for Doslidzhennya operatsiy, LR 6 variant 91");  
 LinearFractionalProblem min = new LinearFractionalProblem(  
 new Double[]{3.0, 4.0}, new Double[]{2.0, 1.0});  
 min.addFunction(new Function(new Double[]{8.0, -5.0}, 40, Function.Sign.*LESS*));  
 min.addFunction(new Function(new Double[]{2.0, 5.0}, 10, Function.Sign.*GREATER*));  
 min.addFunction(new Function(new Double[]{-6.0, 5.0}, 60, Function.Sign.*LESS*));  
 min.addFunction(new Function(new Double[]{2.0, 1.0}, 14, Function.Sign.*LESS*));  
  
 FunctionSystem max = new LinearFractionalProblem(min);  
 System.*out*.println(min);  
  
 max.getOptimal(true);  
 min.getOptimal(false);  
 }  
}

**Результат виконання програми**







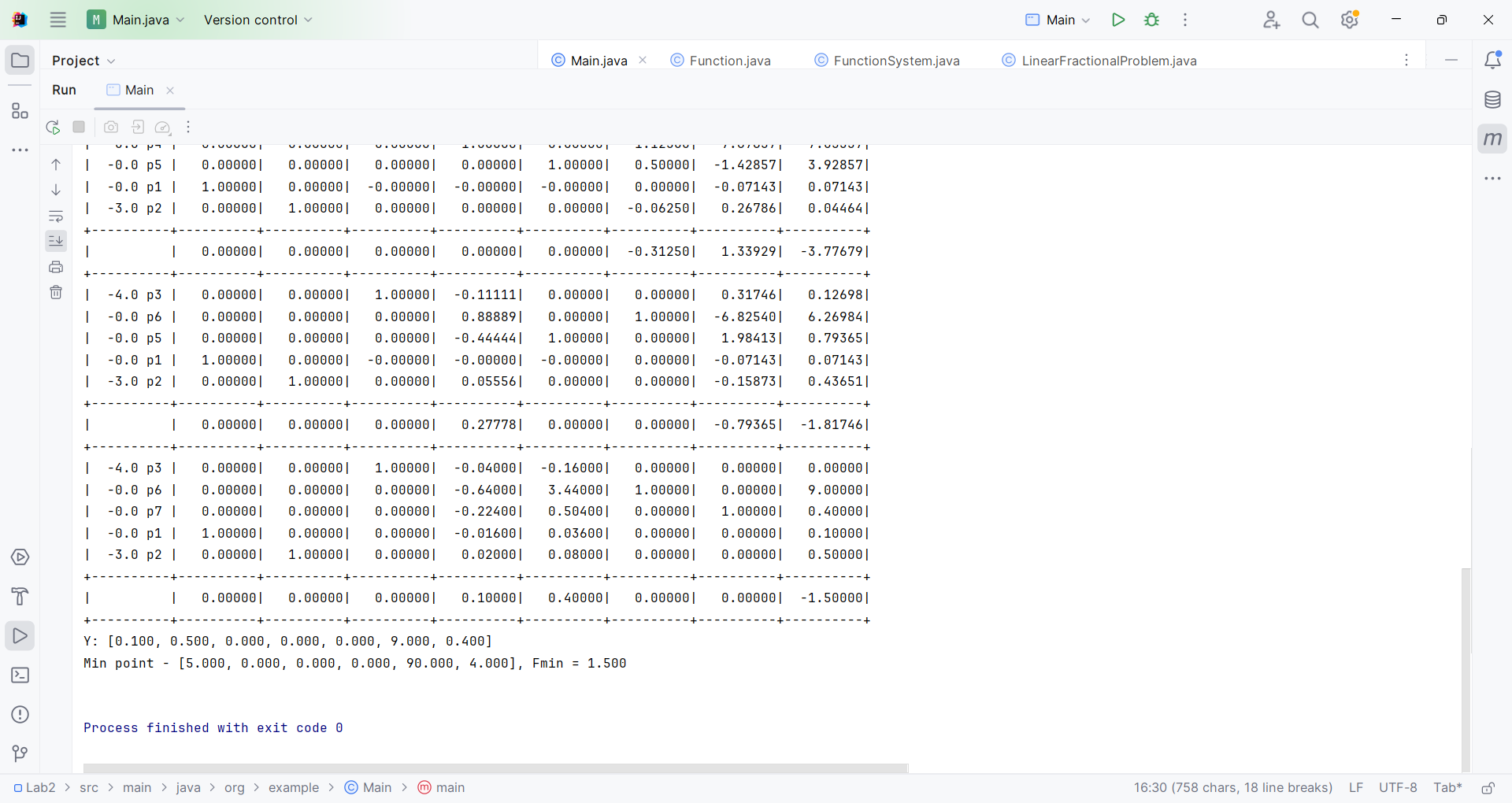


Рис. 6 Результат виконання програми.

**Висновки**

Згідно результатів та порівняння отриманих під час виконання програми – алгоритм виконується правильно та без помилок.

Було знайдено розв’язок за допомогою графічного методу з використанням Excel для максимума і мінімума.

Було знайдено розв’язок аналітично для максимума і мінімума, який співпав з розв’язком знайденого за допомогою графічного методу.

Було розроблено програму з використанням мови Java, яка аналогічно знайшла максимум і мінімум, який відповідає минулим результатам. Кожний клас грає певну роль і розміщується у відповідному файлі під назвою, у цих файлах є коментар, який пояснює роботу класу та його методів.

Було розроблено програмний код. Програму було виконано у середовищі розробки Idea на мові програмування Java з версією 17 STL.