Министерство науки и высшего

образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Рыбинский государственный авиационный технический университет  
имени П. А. Соловьева»

Кафедра математического и программного обеспеченияэлектронных вычислительных средств  
  
  
ЛАБОРАТОРНАЯ РАБОТА №4  
  
по дисциплине  
Исследование операций  
на тему  
««Целочисленное линейное программирование»Вариант №2  
  
Студент группы ИПБ-22\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Новиков А. С.  
Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Задорина Н. А.  
  
  
  
Рыбинск 2024

Задание

Решить задачу целочисленного линейного программирования методом ветвей и границ

Изображение выглядит как текст, Шрифт, рукописный текст, чек

Автоматически созданное описание

**Результат работы программы**

Изображение выглядит как текст, снимок экрана, документ, чек

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, документ, чек

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, документ, чек

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, документ

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание

**Код программы**

int main()

{

    TMath math;

    std::vector<std::vector<double>> matrix = {

        {6, 9},

        {2, 1},

        {1, 0},

        {0, 1}};

    std::vector<double> rightSide = {41, 13, 0, 0};

    std::vector<int> signs = {1, 1, -1, -1};

    std::vector<double> func = {5, 7};

    bool toMax = true;

    if (!math.loadMatrix(matrix, signs, rightSide, func, toMax)){

        output << "Ошибка при создании матрицы!\n";

        return 0;

    }

    math.printSystem(true);

    auto result = math.branchAndBoundaryMethodV(true, false, false);

    if (result.size() == 0){

        output << "Решение не найдено!\n";

        return 0;

    }

    output << "\nЛучшее целочисленное решение:\n";

    for (int i = 0; i < math.getNonBaseVarsCount(); i++)

        output << "X" << i + 1 << "= " << result[i] << std::endl;

    for (int i = 0; i < result.size() - math.getNonBaseVarsCount(); i++)

        func.push\_back(0);

    output << "F = " << math.solveFunc(func, result) << std::endl;

}

bool toMax;

    std::vector<double> optimal;

    std::vector<int> baseVarsPlace;

    int nonbaseVarCount = 0;

 void separateBranch(std::vector<std::vector<double>> table, std::string curBranch, bool findMax, bool visible, bool visibleSimplex, bool hideBaseVars) {

        if (visible)  {

            outFile << "\n==Ветка " << curBranch << "==\n\n";

            printSystem(table, hideBaseVars);

        }

        std::vector<double> values;

        if (visibleSimplex)    values = solveSimplex(table, true);

        else   values = solveSimplex(table, false);

        auto f = solveFunc(table[table.size() - 1], values) \* (toMax ? -1 : 1);

        int nonInteger = -1;

        for (int i = 0; i < values.size() - 1; i++)

            if (!isInteger(values[i]))  {

                nonInteger = i;

                break;

            }

        if (nonInteger == -1)  {

            if (visible)  {

                for (int i = 0; i < nonbaseVarCount; i++)

                    outFile << "x" << (i + 1) << " = " << values[i] << std::endl;

                outFile << "F=" << f << std::endl;

                outFile << "Целочисленное решение.\n";

            }

            auto cur = solveFunc(table[table.size() - 1], optimal) \* (toMax ? -1 : 1);

            if (!optimal.size())  {

                for (int i = 0; i < nonbaseVarCount; i++)

                    optimal.push\_back(values[i]);

                if (visible)     outFile << "Новое значение лучше предыдущего. Сохраняем.\n";

            }

            else if ((f > cur) == findMax)   {

                for (int i = 0; i < nonbaseVarCount; i++)

                    optimal[i] = values[i];

                if (visible)

                    outFile << "Новое значение лучше предыдущего. Сохраняем.\n";

            }

        }

        else

        {

            if (table[0][nonInteger] && ((ceil(values[nonInteger]) == table[0].back() && table[0][table[0].size() - 2] == -1) || (ceil(values[nonInteger]) - 1 == table[0].back() && table[0][table[0].size() - 2] == 1)))            {

                if (visible)

                    outFile << "Нет решения!\n";

                return;

            }

            std::vector<double> temp = std::vector<double>(table[0].size() - 1, 0);

            for (int i = 0; i < table.size(); i++)            {

                table[i].push\_back(table[i][table[i].size() - 1]);

                table[i][table[i].size() - 2] = 0;

            }

            temp[nonInteger] = 1;

            temp.push\_back(-1);

            temp.push\_back(ceil(values[nonInteger]));

            if (visible)            {

                for (int i = 0; i < nonbaseVarCount; i++)

                    outFile << "x" << (i + 1) << " = " << values[i] << std::endl;

                outFile << "F=" << f << std::endl;

                outFile << "Нецелочисленное решение. Разветвляемся\n";

            }

            table.insert(table.begin(), temp);

            baseVarsPlace.insert(baseVarsPlace.begin(), (temp.size() - 2) \* temp[temp.size() - 2]);

            separateBranch(table, curBranch + '1', findMax, visible, visibleSimplex, hideBaseVars);

            if (visible)        outFile << "\n==Откат в ветка " << curBranch << "==\n";

            table[0][table[0].size() - 2] = 1;

            table[0].back()--;

            baseVarsPlace[0] \*= -1;

            separateBranch(table, curBranch + '2', findMax, visible, visibleSimplex, hideBaseVars);

            table.erase(table.begin());

            baseVarsPlace.erase(baseVarsPlace.begin());

        }

    }

std::vector<double> branchAndBoundaryMethodV(bool findMax, bool visibleSimplex, bool visibleBaseVars = false)   {

        if (data.size()) {

            separateBranch(data, "1", findMax, true, visibleSimplex, !visibleBaseVars);

            return optimal;

        }

        outFile << "Пустая матрица\n";

        return {};

    }

**Вывод**

В ходе выполнения лабораторной работы был успешно применен метод ветвей и границ для решения задачи целочисленного программирования. Совпадение результатов вычислений разными способами подтвердило правильность выполненных расчётов.