МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 2**

по дисциплине: Исследование операций

тема: «Симплекс-метод в чистом виде»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

проф. Вирченко Юрий Петрович

Белгород 2024г.

**Лабораторная работа №2**

**«Симплекс-метод в чистом виде»**

*Цель работы:*изучение симплекс-метода для решения задачи линейного программирования с использованием симплекс-таблиц, получение навыков кодирования изученного алгоритма, отладки и тестирования соответствующих программ.

Вариант 3

**Ход выполнения лабораторной работы:**

Запрограммировать и отладить изученный алгоритм.

*Блок-схемы*

Изображение выглядит как текст, Шрифт, снимок экрана, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Изображение выглядит как текст, диаграмма, зарисовка, рисунок

Автоматически созданное описание

Изображение выглядит как текст, чек, диаграмма, черно-белый

Автоматически созданное описание

*Исходный код программы:*

Файл task1.tpp

#ifndef OPERATIONS\_RESEARCH\_TASK1\_TPP  
#define OPERATIONS\_RESEARCH\_TASK1\_TPP  
  
#pragma once  
  
#include <optional>  
#include <vector>  
#include <algorithm>  
#include <iomanip>  
  
#include "../lab1/task1.tpp"  
  
template <std::size\_t *T*>  
auto getMatrixForSimplexMethod(std::vector<std::array<double, *T*>>& matrix, std::array<double, *T*>& function) {  
 // Определим, какие переменные могут быть базисными в опорном решении и занесём их в список functionBasisVars  
 // (Если yi = 0, то переменная может быть базисной)  
 std::vector<int> functionBasisVars;  
 for (int i = 0; i < function.size() - 1; i++)  
 if (std::abs(function[i]) < EPS)  
 functionBasisVars.push\_back(i);  
  
 // Получим все базисные решения и для каждого решения basis  
 for (auto& basis : getAllBasises(matrix)) {  
 bool isAllBsMoreOrEqualToZero = true;  
 for (int i = 0; i < basis.matrix.size() && isAllBsMoreOrEqualToZero; i++) {  
 if (basis.matrix[i].back() < EPS)  
 isAllBsMoreOrEqualToZero = false;  
 }  
  
 // Хоть один из br меньше 0?  
 if (!isAllBsMoreOrEqualToZero) {  
 // Перейти к следующему базисному решению  
 continue;  
 }  
  
 // Если список базисных переменных из опорного решения basis входит в список  
 // functionBasisVars  
 std::sort(basis.indices.begin(), basis.indices.end());  
 if (std::includes(basis.indices.begin(), basis.indices.end(), functionBasisVars.begin(), functionBasisVars.end()))  
 // Возвращаем искомый базис  
 return basis;  
 }  
  
 // Возвращаем ошибку - получить подходящее преобразование матрицы невозможно  
 throw std::invalid\_argument("No basis for function found");  
}  
  
template <std::size\_t *T*>  
double solveSimplexMethodMaxRaw(std::vector<std::array<double, *T*>>& matrix, std::array<double, *T*>& function, std::vector<int>\* baseIndices = nullptr) {  
 // Строим симплекс-таблицу, копируя в неё матрицу matrix  
 std::vector<std::array<double, *T*>> simplexMatrix(matrix);  
 // Добавляем новую строку - целевую функцию, умножая её коэф. yi на -1  
 simplexMatrix.push\_back(function);  
 for (int i = 0; i < *T*; i++)  
 simplexMatrix.back()[i] \*= -1;  
  
 // Бесконечный цикл  
 while (true) {  
 for (int i = 0; i < simplexMatrix.size(); i++) {  
 for (int j = 0; j < simplexMatrix[i].size(); j++) {  
 std::cout << std::setw(10) << simplexMatrix[i][j];  
 }  
  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
  
 // Найдём наибольший по модулю отрицательный элемент в последней строке, кроме свободного члена.  
 int minColumnIndex = -1;  
 for (int i = 0; i < *T* - 1; i++) {  
 if (simplexMatrix.back()[i] < 0 && (minColumnIndex == -1 || simplexMatrix.back()[minColumnIndex] > simplexMatrix.back()[i]))  
 minColumnIndex = i;  
 }  
  
 // Такой элемент найден?  
 if (minColumnIndex == -1) {  
 // Решение получено, можно выходить из цикла  
 break;  
 }  
  
 // Определим генеральный элемент таблицы  
 int minRowIndex = -1;  
 for (int i = 0; i < simplexMatrix.size() - 1; i++) {  
 if (simplexMatrix[i][minColumnIndex] <= EPS) continue;  
 if (minRowIndex == -1) minRowIndex = i;  
 else if (simplexMatrix[minRowIndex].back() / simplexMatrix[minRowIndex][minColumnIndex] >  
 simplexMatrix[i].back() / simplexMatrix[i][minColumnIndex]) minRowIndex = i;  
 }  
  
 // Такой элемент найден?  
 if (minRowIndex == -1) {  
 // Решение получить невозможно, возвращаем ошибку  
 throw std::invalid\_argument("No solution");  
 }  
  
 if (baseIndices)  
 (\*baseIndices)[minRowIndex] = minColumnIndex;  
  
 // Преобразуем матрицу к новому базисному виду  
 subtractLineFromOther(simplexMatrix, minRowIndex, minColumnIndex);  
 }  
  
 for (int i = 0; i < simplexMatrix.size(); i++) {  
 for (int j = 0; j < simplexMatrix[i].size(); j++) {  
 std::cout << std::setw(10) << simplexMatrix[i][j];  
 }  
  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
  
 for (int i = 0; i < matrix.size(); i++)  
 for (int j = 0; j < matrix[i].size(); j++)  
 matrix[i][j] = simplexMatrix[i][j];  
  
 // Возвращаем свободный член в последней строке  
 std::cout << "Zmax: ";  
 return {simplexMatrix.back().back()};  
}  
  
template <std::size\_t *T*>  
double solveSimplexMethodMax(std::vector<std::array<double, *T*>>& matrix, std::array<double, *T*>& function) {  
 // Подготовим матрицу таким образом, чтобы в целевой функции были использованы только свободные переменные.  
 // Также отберём опорное решение, в котором все свободные члены больше или равны нулю  
 auto preparedMatrix = getMatrixForSimplexMethod(matrix, function);  
  
 // Вызовем симплекс метод на преобразованной матрице  
 return solveSimplexMethodMaxRaw(preparedMatrix.matrix, function);  
}  
  
#endif //OPERATIONS\_RESEARCH\_TASK1\_TPP

Файл task1.cpp

#include <iostream>  
#include <iomanip>  
#include <array>  
  
#include "../../libs/alg/alg.h"  
  
int main() {  
 // Инициализируем матрицу и функцию  
 std::vector<std::array<double, 7>> matrix = **{** {1, -3, 0, -4, 0, -2, 10},  
 {0, 5, 0, 5, 1, 1, 26},  
 {0, 4, 1, -6, 0, -3, 12}**}**;  
 std::array<double, 7> function{{0.0, 7.0, 0.0, 8.0, 0.0, 1.0, 0.0}};  
  
 // Вывод ответа  
 std::cout << "Zmax: " << solveSimplexMethodMax(matrix, function);  
  
}

Результат работы программы:



Дополнительно проведём тестирование при помощи программы отбора базисных решений (см. лабораторная работа №1 задание 2).

Результат выполнения программы:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Результаты вычислений:**

Имеем целевую функцию . И систему уравнений

Приведём систему уравнений к базисному виду. Базисными переменными выберем .

Привели систему к базисному виду

Построим симплекс- таблицу

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Баз. Пер. | Св. чл. |  |  |  |  |  |  |
|  |  | 1 | -3 | 0 | -4 | 0 | -2 |
|  |  | 0 | 4 | 1 | -6 | 0 | -3 |
|  | 26 | 0 | 5 | 0 | 5 | 1 | 1 |
|  | 0 | 0 | -7 | 0 | -8 | 0 | -1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Баз. Пер. | Св. чл. |  |  |  |  |  |  |
|  |  | 1 | 1 | 0 | 0 |  |  |
|  |  | 0 | 10 | 1 | 0 |  |  |
|  |  | 0 | 1 | 0 | 1 |  |  |
|  |  | 0 | 1 | 0 | 0 |  |  |

Ответ: . Координаты точки максимума: . Результаты ручных вычислений совпали с результатами выполнения программы.

**Вывод:** в ходе лабораторной работы разработали и отладили программу, находящую оптимальное решение в системе линейных уравнений для целевой функции, и использующей симплекс метод.