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 № 1**

по дисциплине: Исследование операций

тема: «Исследование множества опорных планов системы ограничений задачи линейного программирования (задачи ЛП) в канонической форме»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

проф. Вирченко Юрий Петрович

Белгород 2024г.

**Лабораторная работа №1**

**«Исследование множества опорных планов системы ограничений задачи линейного программирования (задачи ЛП) в канонической форме»**

*Цель работы:*изучить метод Гаусса-Жордана и операцию замещения, а также освоить их применение к отысканию множества допустимых базисных видов системы линейных уравнений, и решению задачи линейного программирования простым перебором опорных решений.

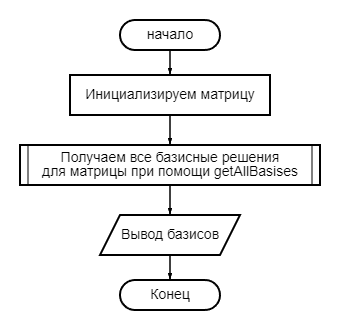
Вариант 3

****

**Ход выполнения лабораторной работы:**

1. Составить программу для отыскания всех базисных видов системы линейных уравнений.

*Блок-схемы*

**

Изображение выглядит как текст, диаграмма, зарисовка, рисунок

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, чек

Автоматически созданное описание

*Исходный код программы:*

Файл task1.tpp

#pragma once  
  
#include <vector>  
#include <cstdlib>  
#include <stdexcept>  
  
#define EPS 0.00000000001  
  
// Сочетания  
template<typename T>  
std::vector<std::vector<T>> getCombinations(std::vector<T> &baseSet, std::vector<T> currentSet, size\_t minIndex, size\_t k, size\_t count) {  
 std::vector<std::vector<T>> resultCombs;  
  
 // Если количество перестановок равно необходимому, мы достигли искомого множества, возвращаем его  
 if (count >= k)  
 return {currentSet};  
  
 for (size\_t i = minIndex; i <= baseSet.size() - k + count; i++) {  
 // Добавляем в текущее множество новый элемент, Ci = x  
 std::vector<T> newCurrentSet(currentSet);  
 newCurrentSet.push\_back(baseSet[i]);  
  
 // Вызываем следующий шаг итерации, сохраняем его результат в общий массив множеств  
 std::vector<std::vector<T>> combinations = getCombinations(baseSet, newCurrentSet, i + 1, k, count + 1);  
 resultCombs.insert(std::begin(resultCombs), std::begin(combinations), std::end(combinations));  
 }  
  
 // Возвращаем массив множеств  
 return resultCombs;  
}  
  
// Функция-обёртка для рекуррентной функции  
template<typename T> std::vector<std::vector<T>> getCombinations(std::vector<T> &baseSet, size\_t k) {  
 return getCombinations(baseSet, {}, 0, k, 0);  
}  
  
// Перестановка  
template<typename T>  
std::vector<std::vector<T>> getPermutations(std::vector<T> baseSet, std::vector<T> currentSet) {  
 std::vector<std::vector<T>> resultPerms;  
  
 // Если элементов в изначальном множестве не осталось, получено искомое множество  
 if (baseSet.size() == 0) return {currentSet};  
  
 for (size\_t i = 0; i < baseSet.size(); i++) {  
 // Удаляем из исходного массива x  
 std::vector<T> newBaseSet(baseSet);  
 newBaseSet.erase(std::begin(newBaseSet) + i);  
  
 // Добавляем в текущее множество новый элемент  
 std::vector<T> newCurrentSet(currentSet);  
 newCurrentSet.push\_back(baseSet[i]);  
  
 // Выполняем следующий шаг итерации, сохраняем в итоговый массив множеств  
 auto permutations = getPermutations(newBaseSet, newCurrentSet);  
 resultPerms.insert(std::begin(resultPerms), std::begin(permutations), std::end(permutations));  
 }  
  
 return resultPerms;  
}  
  
template<typename T>  
std::vector<std::vector<T>> getPermutations(std::vector<T> &baseSet) {  
 return getPermutations(baseSet, {});  
}  
  
template <std::size\_t *T*>  
void subtractLineFromOther(std::vector<std::array<double, *T*>>& origin, int indexLeadingLine, int indexEnablingElement) {  
 // Преобразуем ведущую строку таким образом, чтобы разрещающий элемент стал равен 1.  
 double originEnablingElement = origin[indexLeadingLine][indexEnablingElement];  
 for (int i = 0; i < origin[indexLeadingLine].size(); i++)  
 // Разделим каждый элемент строки по индексу indexLeadingLine  
 // на разрешающий элемент  
 origin[indexLeadingLine][i] /= originEnablingElement;  
  
 // Для всех остальных строк с номером i  
 for (int i = 0; i < origin.size(); i++) {  
 if (i == indexLeadingLine) continue;  
  
 // k - коэффициент, на который нужно домножить ведущую строку и вычесть её  
 double k = origin[i][indexEnablingElement];  
  
 // Для всех остальных элементов j в строке i  
 for (int j = 0; j < origin[indexLeadingLine].size(); j++)  
 // Вычесть ведущую строку indexLeadingLine из строки i  
 origin[i][j] -= origin[indexLeadingLine][j] \* k;  
 }  
}  
  
// Вспомогательная СД  
template <std::size\_t *T*>  
struct Basis {  
 std::vector<int> indices;  
 std::vector<std::array<double, *T*>> matrix;  
};  
  
template <std::size\_t *T*>  
std::vector<Basis<*T*>> getAllBasises(std::vector<std::array<double, *T*>> origin) {  
 // result - массив полученных систем  
 std::vector<Basis<*T*>> result;  
  
 // indices - массив неизвестных в системе, заполняем  
 // его индексами 0 ... l - 1, где l - длина  
 // матрицы origin  
 std::vector<int> indices;  
 for (int i = 0; i < origin[0].size() - 1; i++)  
 indices.push\_back(i);  
  
 /\*  
 Для каждого сочетания в indices из l по h (h - высота матрицы)  
 получим набор базисных неизвестных basis.  
 \*/  
 for (auto basis : getCombinations(indices, origin.size())) {  
 // Для каждой перестановки строк в матрице origin matrixPermutation  
 for (auto matrixPermutation : getPermutations(origin)) {  
 auto copyMatrixPermutation = matrixPermutation;  
 bool badPermutation = false;  
  
 // Для каждой строки в матрице (ввод счётчика строк i)  
 for (int i = 0; i < matrixPermutation.size(); i++) {  
 // Разрешающий элемент равен 0?  
 if (std::abs(matrixPermutation[i][basis[i]]) < EPS) {  
 bool allZeros = true;  
 for (int j = 0; (j < matrixPermutation[i].size() - 1) && allZeros; j++) {  
 if (std::abs(matrixPermutation[i][j]) > EPS) {  
 // Неудачное расположение строк, необходимо  
 // перейти к другой перестановке  
 badPermutation = true;  
 allZeros = false;  
 break;  
 }  
 }  
  
 // Коэффициенты в i строке содержат только 0?  
 if (!allZeros) {  
 // Перейти к следующей перестановке строк  
 break;  
 }  
  
 // b\_i равен 0?  
 if (std::abs(matrixPermutation[i].back()) > EPS) {  
 // Система несовместима. Вернуть пустой массив.  
 return {};  
 }  
  
 /\*  
 Удалить строку i из матрицы, полученной перестановкой строк,  
 и вызвать функцию getAllBasises с полученной матрицей.  
 \*/  
 if (allZeros) {  
 copyMatrixPermutation.erase(copyMatrixPermutation.begin() + i);  
 return getAllBasises(copyMatrixPermutation);  
 }  
  
 break;  
 }  
  
 // Выберем ведущую переменную из basis[i], преобразуем i строку и вычтем её из остальных  
 subtractLineFromOther(matrixPermutation, i, basis[i]);  
 }  
  
 if (badPermutation) continue;  
  
 // Полученную матрицу matrixPermutation добавим в result, закончим перебор перестановок  
 result.push\_back({basis, matrixPermutation});  
 break;  
 }  
 }  
  
 return result;  
}

Файл task1.cpp

#include <iostream>  
#include <iomanip>  
#include <array>  
#include <windows.h>  
  
#include "../../libs/alg/labs/lab1/task1.tpp"  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
 // Инициализируем матрицу  
 std::vector<std::array<double, 6>> matrix = **{** { 2, -1, 6, -1, 3, 12},  
 { 3, 5, 1, -12, 2, 14},  
 {-3, 6, 8, 7, -4, 18}**}**;;  
  
 // Получаем базисные решения  
 auto res = getAllBasises(matrix);  
  
 // Выводим базисные решения  
 std::cout << "==================================================================================================================\n";  
 for (auto& matrix : res) {  
 std::cout << "Выбранные базисные переменные: ";  
 for (auto& bas : matrix.indices) {  
 std::cout << "x" << (bas + 1) << " ";  
 }  
 std::cout << "\n\nПолученная система: " << std::endl;  
  
  
 for (int i = 0; i < matrix.matrix[0].size() - 1; i++) {  
 std::stringstream buf;  
 buf << "a" << (i + 1);  
 std::cout << std::setw(15) << buf.str() << " ";  
 }  
  
 std::cout << std::setw(15) << "b" << std::endl;  
  
 for (auto & line : matrix.matrix) {  
 for (auto & element : line) {  
 std::cout << std::setw(15) << element << " ";  
 }  
  
 std::cout << "\n";  
 }  
  
 std::cout << "==================================================================================================================\n";  
 }  
}

Результат работы программы:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

1. Организовать отбор опорных планов среди всех базисных решений, а также нахождение оптимального опорного плана методом прямого перебора. Целевая функция выбирается произвольно.

*Блок-схемы*

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

Автоматически созданное описание

*Исходный код программы:*

#include <iostream>  
#include <iomanip>  
#include <windows.h>  
#include <limits>  
#include <algorithm>  
#include <array>  
  
#include "../../libs/alg/labs/lab1/task1.tpp"  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
 // Инициализируем матрицу  
 std::vector<std::array<double, 6>> matrix = **{** { 2, -1, 6, -1, 3, 12},  
 { 3, 5, 1, -12, 2, 14},  
 {-3, 6, 8, 7, -4, 18}**}**;  
  
 // Получаем все базисные решения для матрицы при помощи getAllBasises  
 auto res = getAllBasises(matrix);  
  
 // Вводим переменные для хранения максимального значения функции и оптимального плана решения   
 double maxFunctionValue = std::numeric\_limits<double>::min();  
 std::vector<double> maxB;  
 // Задаём функцию z = -10 \* x1 + 2 \* x2 + 75 \* x3 - 18 \* x6  
 std::vector<double> function = **{**-10, 2, 75, 0, 0, -18**}**;  
 std::cout << "==================================================================================================================\n";  
  
 // Для всех полученных базисов  
 for (auto &basis : res) {  
 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;  
 }  
  
 double z = 0;  
 std::vector<double> B;  
 // Для каждого коэффициента в функции z  
 for (int i = 0; i < function.size(); i++) {  
 // Базисная переменная есть в решении?  
 if (std::find(basis.indices.begin(), basis.indices.end(), i) != basis.indices.end()) {  
 for (int j = 0; j < basis.matrix.size(); j++) {  
 if (std::abs(basis.matrix[j][i] - 1.0) < EPS) {  
 // Прибавляем значение br, умноженное на коэффициент в функции  
 z += function[i] \* basis.matrix[j].back();  
 B.push\_back(basis.matrix[j].back());  
 break;  
 }  
 }  
 } else {  
 // Переменная свободная, следовательно равна 0  
 B.push\_back(0);  
 }  
 }  
  
 // Выводим найденное опорное решение и значение функции z   
 std::cout << "Обнаружено опорное решение: {";  
 for (int i = 0; i < B.size(); i++) {  
 std::cout << B[i] << "; ";  
 }  
 std::cout << "\b\b}\n\nЗначение функции z(B): " << z << "\n";  
  
 std::cout << "==================================================================================================================" << std::endl;  
  
 // Полученное значение функции больше сохранённой?  
 if (z > maxFunctionValue) {  
 // Запоминаем его и базисное решение  
 maxFunctionValue = z;  
 maxB = B;  
 }  
 }  
  
 // Выводим значение z и базисное решение  
 std::cout << "\nZmax: " << maxFunctionValue << "\n\nОптимальный план: {";  
 for (int i = 0; i < maxB.size(); i++) {  
 std::cout << maxB[i] << "; ";  
 }  
 std::cout << "\b\b}" << std::endl;  
}

Результат работы программы.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Решить одну из следующих ниже задач вручную (подготовить тестовые данные).

Исходная система уравнений:

Построение расширенной матрицы:

2 -1 6 -1 3 ∣ 12

3 5 1 -12 2 ∣ 14

-3 6 8 7 -4 ∣ 18

Вычтем из второй строки первую, умноженную на 1.5.

Вычтем из третьей строки первую, умноженную на -1.5.

2 -1 6 -1 3 ∣ 12

0 7.5 -8 -13.5 -2.5 ∣ -6

0 12 17 1 5.5 -0.5 ∣ 36

Вычтем из третьей строки вторую.

2 -1 6 -1 3 ∣ 12

0 1 -1.066 -1.8 -0.333 ∣ -0.8

0 0 2.482 3.092 -0.291 ∣ 3.8

Приведем второй ведущий элемент к 1 путем деления строки на 1.

Вычтем из первой строки шесть вторых строк, умноженных на 1.

Вычтем из третьей строки две вторых строки, умноженных на 2.482.

2 0 0 10 5 ∣ 16

0 1 -1.066 -1.8 -0.333 ∣ -0.8

0 0 1 1.246 -0.117 ∣ 1.532

Приведем третий ведущий элемент к 0 путем вычитания второй строки, умноженной на 1.246.

Вычтем из первой строки 1.246 третьей строки.

2 0 0 8.708 5.291 ∣ 14.468

0 1 0 -3.054 -0.463 ∣ -1.422

0 0 1 1.246 -0.117 ∣ 1.52

Теперь можем выразить базисные переменные через свободные переменные:

**Вывод:** в ходе лабораторной работы разработали и отладили программу, находящую базисные решения системы уравнений методом Гаусса-Жордана.