МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Системное программирование»

Лабораторная работа №1

**«Графическое решение ЗЛП»**

по дисциплине

Теория систем и системный анализ

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Полищук Ю.В.

Москва, 2024 г

Содержание

[**1.** **Лабораторная работа №1** 3](#_Toc177284701)

[**1.1.** **Цель работы** 3](#_Toc177284702)

[**1.2.** **Задание** 3](#_Toc177284703)

[**1.3.** **Ход работы** 3](#_Toc177284704)

[**1.4.** **Заключение** 17](#_Toc177284705)

1. **Лабораторная работа №1**
   1. **Цель работы**

Изучить методы решения задачи линейного программирования графическим методом.

* 1. **Задание**

Решить 4 задания линейного программирования. Показать результаты решения, сверить ответы

* 1. **Ход работы**

**Теоретическая информация**

Решить Линейные уравнения, задающие задачи можно решить несколькими способами. Одним из самых простых является графический метод. Суть графического метода заключается в построение на графике уравнений, выделения конечных точек и последующий поиск минимума или максимума

**Практическая реализация**

Напишу программу на языке C++, использовав лишь 4 библитеки: «iomanip» и «iostream» – для вывода данных в консоль, «cmath» – для вызова некоторых математических функций, «SFML» - для создания графического интерфейса. Перепроверю результаты программы с результатами, полученными с онлайн-решателя.

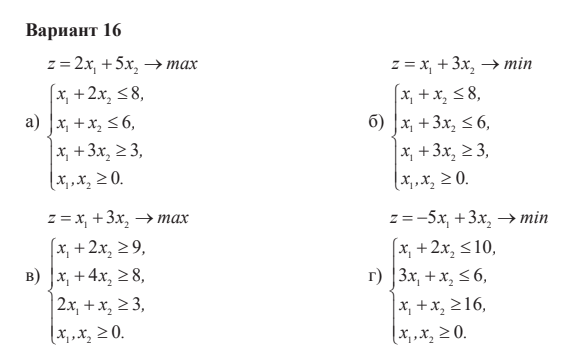


Рисунок 1 – условие четырёх индивидуальных заданий

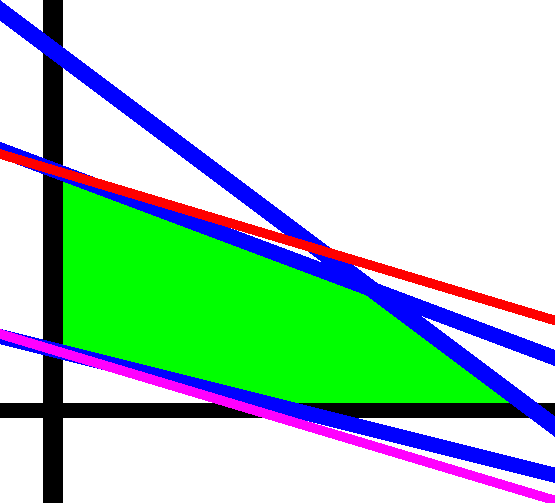


Рисунок 2 – графический результат работы программы на примере 1



Рисунок 3 – результат работы программы на примере 1

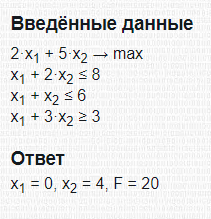


Рисунок 4 – проверка ответа

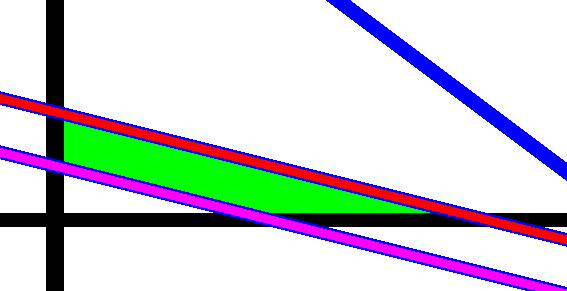


Рисунок 5 – графический результат работы программы на примере 2



Рисунок 6 – результат работы программы на примере 2

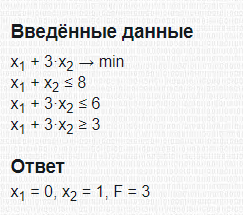


Рисунок 7 – проверка ответа

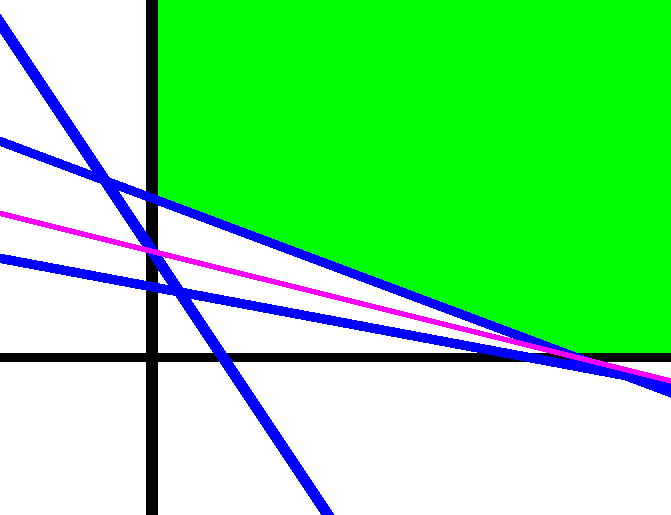


Рисунок 8 – графический результат работы программы на примере 3



Рисунок 9 – результат работы программы на примере 3

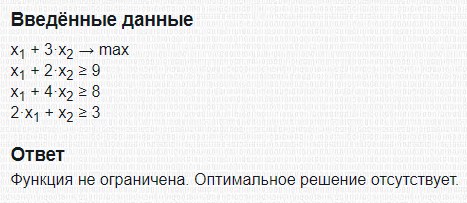


Рисунок 10 – проверка ответа

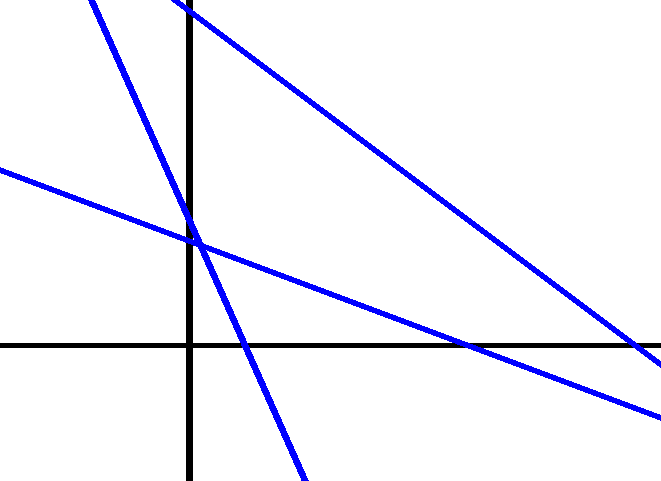


Рисунок 11 – графический результат работы программы на примере 4



Рисунок 12 – результат работы программы на примере 4

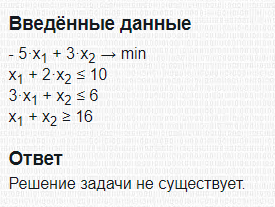


Рисунок 13 – проверка ответа

Программный код представлен ниже.

#include <SFML/Graphics.hpp>

#include <cmath>

#include <iostream>

#include <iomanip>

**using** **namespace** std;

**class** **sfLine** : **public** sf::Drawable

{

**public:**

sfLine(**const** sf::Vector2f& point1, **const** sf::Vector2f& point2, sf::Color color\_ = sf::Color::Black, **float** thick = **5.f**):

color(color\_), thickness(thick)

{

sf::Vector2f direction = point2 - point1;

sf::Vector2f unitDirection = direction/std::sqrt(direction.x\*direction.x+direction.y\*direction.y);

sf::Vector2f unitPerpendicular(-unitDirection.y,unitDirection.x);

sf::Vector2f offset = (thickness/**2.f**)\*unitPerpendicular;

vertices[**0**].position = point1 + offset;

vertices[**1**].position = point2 + offset;

vertices[**2**].position = point2 - offset;

vertices[**3**].position = point1 - offset;

**for** (**int** i=**0**; i<**4**; ++i)

vertices[i].color = color;

}

**void** draw(sf::RenderTarget &target, sf::RenderStates states) **const**

{

target.draw(vertices,**4**,sf::Quads);

}

**public:**

sf::Vertex vertices[**4**];

**float** thickness;

sf::Color color;

};

/// ax + by + c = 0 <=> sfLine (-y, т.к. в sfml Oy перевёрнута)

pair<sf::Vector2f, sf::Vector2f> get\_Line\_from\_equ(**long** **double** a, **long** **double** b, **long** **double** c){

**long** **double** pseudo\_inf = **1e5**;

**if** (abs(a) < **0.0001** ){

**if** (abs(b) < **0.0001**){

std::cerr << "Equation cannot be a form of 0x + 0y";

exit(-**1**);

}

**long** **double** y\_1 = (-a\*pseudo\_inf - c) / b;

**long** **double** y\_2 = (-a\*(-pseudo\_inf) - c) / b;

**return** {{pseudo\_inf, -y\_1},{-pseudo\_inf, -y\_2,}};

}

**long** **double** x\_1 = (-b\*pseudo\_inf - c) / a;

**long** **double** x\_2 = (-b\*(-pseudo\_inf) - c) / a;

**return** {{x\_1,-pseudo\_inf},{x\_2,-(-pseudo\_inf)}};

}

vector<vector<**long** **double**>> equations\_input;

vector<**long** **double**> target\_function; /// 2x1 + 5x2 -> max

sfLine **get\_line**(**const** vector<**long** **double**>& equation, sf::Color clr = sf::Color::Red){

**auto** p = get\_Line\_from\_equ(equation[**0**], equation[**1**], -equation[**2**]);

**return** sfLine (p.first, p.second, clr, **0.25**); /// 2x1 + 5x2 -> max <=>

}

sfLine **get\_contour\_line**(**double** F\_value, sf::Color clr = sf::Color::Red){

**auto** p = get\_Line\_from\_equ(target\_function[**0**], target\_function[**1**], -F\_value);

**return** sfLine (p.first, p.second, clr, **0.15**); /// 2x1 + 5x2 -> max <=>

}

**double** **dist**(**double** x1, **double** y1, **double** x2, **double** y2){

**return** sqrt((x1 - x2)\*(x1 - x2) + (y1 - y2)\*(y1 - y2));

}

// LINE/LINE

pair<**bool**, sf::Vector2f> lineLine(**float** x1, **float** y1, **float** x2, **float** y2, **float** x3, **float** y3, **float** x4, **float** y4) {

// calculate the direction of the lines

**float** uA = ((x4-x3)\*(y1-y3) - (y4-y3)\*(x1-x3)) / ((y4-y3)\*(x2-x1) - (x4-x3)\*(y2-y1));

**float** uB = ((x2-x1)\*(y1-y3) - (y2-y1)\*(x1-x3)) / ((y4-y3)\*(x2-x1) - (x4-x3)\*(y2-y1));

// if uA and uB are between 0-1, lines are colliding

**if** (uA >= **0** && uA <= **1** && uB >= **0** && uB <= **1**) {

// optionally, draw a circle where the lines meet

**float** intersectionX = x1 + (uA \* (x2-x1));

**float** intersectionY = y1 + (uA \* (y2-y1));

**return** {true, {intersectionX, intersectionY}};

}

**return** {false, {-**1**,-**1**}};

}

**struct** geom\_line{

**long** **double** x1, y1, x2, y2;

};

**long** **double** min\_ = **1e10**; sf::Vector2f min\_point = {-**1**,-**1**};

**long** **double** max\_ = -**1e10**; sf::Vector2f max\_point = {-**1**,-**1**};

vector<sf::Vector2f> convex\_points;

**bool** **check\_point**(**const** sf::Vector2f& point){

**long** **double** value = point.x \* target\_function[**0**] + point.y \* target\_function[**1**];

**for**(**auto** equation : equations\_input){

**long** **double** check\_value = point.x \* equation[**0**] + point.y \* equation[**1**];

**if** (!(check\_value - **0.001** <= equation[**2**]) || point.x < **0** || point.y < **0** ){

**return** false;

}

}

**if** (value < min\_){

min\_ = value;

min\_point = point;

}

**if** (value > max\_){

max\_ = value;

max\_point = point;

}

convex\_points.push\_back(point);

**return** true;

}

**int** **orientation**(sf::Vector2f a, sf::Vector2f b, sf::Vector2f c) {

**double** v = a.x\*(b.y-c.y)+b.x\*(c.y-a.y)+c.x\*(a.y-b.y);

**if** (v < **0**) **return** -**1**; // clockwise

**if** (v > **0**) **return** +**1**; // counter-clockwise

**return** **0**;

}

**bool** **cw**(sf::Vector2f a, sf::Vector2f b, sf::Vector2f c, **bool** include\_collinear) {

**int** o = orientation(a, b, c);

**return** o < **0** || (include\_collinear && o == **0**);

}

**bool** **collinear**(sf::Vector2f a, sf::Vector2f b, sf::Vector2f c) { **return** orientation(a, b, c) == **0**; }

**void** **convex\_hull**(vector<sf::Vector2f>& a, **bool** include\_collinear = false) {

sf::Vector2f p0 = \*min\_element(a.begin(), a.end(), [](sf::Vector2f a, sf::Vector2f b) {

**return** make\_pair(a.y, a.x) < make\_pair(b.y, b.x);

});

sort(a.begin(), a.end(), [&p0](**const** sf::Vector2f& a, **const** sf::Vector2f& b) {

**int** o = orientation(p0, a, b);

**if** (o == **0**)

**return** (p0.x-a.x)\*(p0.x-a.x) + (p0.y-a.y)\*(p0.y-a.y)

< (p0.x-b.x)\*(p0.x-b.x) + (p0.y-b.y)\*(p0.y-b.y);

**return** o < **0**;

});

**if** (include\_collinear) {

**int** i = (**int**)a.size()-**1**;

**while** (i >= **0** && collinear(p0, a[i], a.back())) i--;

reverse(a.begin()+i+**1**, a.end());

}

vector<sf::Vector2f> st;

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

**while** (st.size() > **1** && !cw(st[st.size()-**2**], st.back(), a[i], include\_collinear))

st.pop\_back();

st.push\_back(a[i]);

}

**if** (include\_collinear == false && st.size() == **2** && st[**0**] == st[**1**])

st.pop\_back();

a = st;

}

**int** **main**() {

/\*\*

/// Задание a (Вариант 16)

target\_function = {2,5};

equations\_input.push\_back({1,2,8});

equations\_input.push\_back({1,1,6});

equations\_input.push\_back({-1,-3,-3});

\*/

/\*\*

/// Вариант 16, б

target\_function = {1,3};

equations\_input.push\_back({1,1,8});

equations\_input.push\_back({1,3,6});

equations\_input.push\_back({-1,-3,-3});

\*/

/\*\*

/// Вариант 16, в

target\_function = {1,3};

equations\_input.push\_back({-1,-2,-9});

equations\_input.push\_back({-1,-4,-8});

equations\_input.push\_back({-2,-1,-3});

\*/

/// Вариант 16, г

target\_function = {-**5**,**3**};

equations\_input.push\_back({**1**,**2**,**10**});

equations\_input.push\_back({**3**,**1**,**6**});

equations\_input.push\_back({-**1**,-**1**,-**16**});

vector<geom\_line> lines;

**for**(**auto** equation : equations\_input){

**auto** p = get\_Line\_from\_equ(equation[**0**], equation[**1**], -equation[**2**]);

lines.push\_back({p.first.x, -p.first.y, p.second.x, -p.second.y});

}

sf::RenderWindow window(sf::VideoMode(**800**, **600**), "Plot");

sfLine Oy({**0**,**1e10**}, {**0**,-**1e10**}, sf::Color::Black, **0.25**);

sfLine Ox({**1e10**,**0**}, {-**1e10**,**0**}, sf::Color::Black, **0.25**);

lines.push\_back({**0**,**1e10**,**0**,-**1e10**});

lines.push\_back({**1e10**,**0**,-**1e10**,**0**});

vector<sf::Vector2f> intersec\_points;

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

**for**(**int** j = i + **1**; j < lines.size(); ++j){

**auto** collision\_res = lineLine(lines[i].x1,lines[i].y1,lines[i].x2,lines[i].y2,lines[j].x1,lines[j].y1,lines[j].x2,lines[j].y2);

**if** (collision\_res.first){

intersec\_points.push\_back(collision\_res.second);

}

}

}

**bool** is\_there\_solution = false;

**for**(**auto** point : intersec\_points){

is\_there\_solution |= check\_point(point);

}

**if** (!is\_there\_solution){

cout << "Solution does not exist" << endl;

}

**long** **double** min\_pre = min\_;

**long** **double** max\_pre = max\_;

**for**(**auto** equation : equations\_input){

**auto** p = get\_Line\_from\_equ(equation[**0**], equation[**1**], -equation[**2**]);

check\_point(p.first);

check\_point(p.second);

}

check\_point({**0**,**1e10**});

check\_point({**0**,-**1e10**});

check\_point({**1e10**,**0**});

check\_point({-**1e10**,**0**});

**bool** unbound\_min = false;

**bool** unbound\_max = false;

**if** (min\_pre != min\_){

cout << "Solution isn't bounded to min" << endl;

unbound\_min = true;

}

**if** (max\_pre != max\_){

cout << "Solution isn't bounded to max" << endl;

unbound\_max = true;

}

**if** (!unbound\_min && is\_there\_solution){

cout << fixed << setprecision(**3**) << "F\_min: " << min\_ << " at (" << min\_point.x << " " << min\_point.y << ")" << endl;

}

**if** (!unbound\_max && is\_there\_solution){

cout << fixed << setprecision(**3**) << "F\_max: " << max\_ << " at (" << max\_point.x << " " << max\_point.y << ")" << endl;

}

sf::ConvexShape Polygon;

Polygon.setPointCount(convex\_points.size());

Polygon.setFillColor(sf::Color::Green);

**if** (convex\_points.size() >= **3**){

convex\_hull(convex\_points, false);

Polygon.setPointCount(convex\_points.size());

**int** cou = **0**;

**for**(**auto** point : convex\_points){

Polygon.setPoint(cou++, sf::Vector2f(point.x, -point.y));

}

}

sf::View view(sf::Vector2f(**4.f**, -**4.f**), sf::Vector2f(**10.f**, **10.f**));

window.setView(view);

window.setFramerateLimit(**20**);

**while** (window.isOpen()) {

sf::Event event;

**while** (window.pollEvent(event)) {

**if** (event.type == sf::Event::Closed) {

window.close();

}

**else** **if**(event.type == sf::Event::MouseWheelMoved)

{

**double** scale\_factor = **0.9**;

**for**(**int** i = **0**; i < abs(event.mouseWheel.delta); ++i){

**if** (event.mouseWheel.delta >= **0**)

view.zoom(scale\_factor);

**else**

view.zoom(**1** / scale\_factor);

}

window.setView(view);

}

}

**int** dx = **0**;

**int** dy = **0**;

**if** (sf::Keyboard::isKeyPressed( sf::Keyboard::Key::W)) dy -= **1**;

**if** (sf::Keyboard::isKeyPressed( sf::Keyboard::Key::S)) dy += **1**;

**if** (sf::Keyboard::isKeyPressed( sf::Keyboard::Key::A)) dx -= **1**;

**if** (sf::Keyboard::isKeyPressed( sf::Keyboard::Key::D)) dx += **1**;

view.setCenter(view.getCenter() + sf::Vector2f(dx,dy) \* (view.getSize().x) \* **0.03f** );

window.setView(view);

window.clear(sf::Color::White);

window.draw(Polygon);

window.draw(Ox);

window.draw(Oy);

//window.draw(get\_contour\_line());

**for**(**auto** equation : equations\_input){

window.draw(get\_line(equation, sf::Color::Blue));

}

**if** (!unbound\_max && is\_there\_solution){

window.draw(get\_contour\_line(max\_, sf::Color::Red));

}

**if** (!unbound\_min && is\_there\_solution){

window.draw(get\_contour\_line(min\_, sf::Color::Magenta));

}

window.display();

}

**return** **0**;

}

* 1. **Заключение**

В ходе проделанной работы мы познакомились с графическим методом решения задач линейного программирования