**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

БАЛТИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ВОЕНМЕХ» им. Д.Ф. УСТИНОВА



Дисциплина МЕТОДЫ ОПТИМИЗАЦИИ

практическая работа № 3

|  |
| --- |
| Решение задачи линейного программирования графически |
| и с использованием симплекс-метода |

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил студент | | Анкудинов А.Н. | |
| Фамилия И.О. | | | |
| Группа | А571 | |  |
| Преподаватель | |  | |
|  | | Фамилия И. О. | |

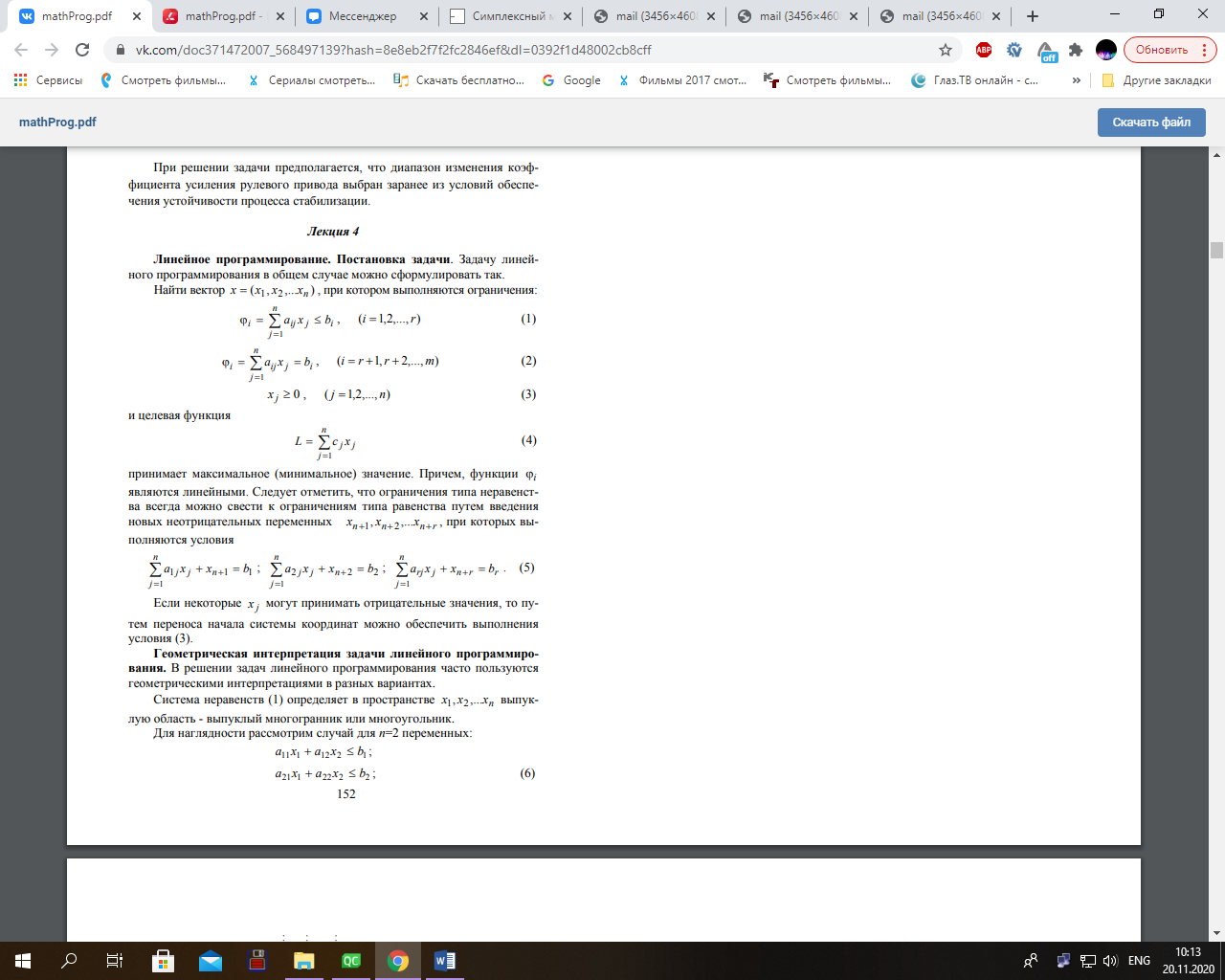
Санкт-Петербург

2020 г.

*Цель работы* – решить задачу линейного программирования графически и с использованием симплекс-метода.Написать программу для решения задачи с помощью симплекс-таблиц.

*Вариант 1. Исходные данные:*

*Постановка задачи* *линейного программирования*



являются линейными.

*Ход работы*

Решение задачи геометрическим способом, симплекс-методом и способом с помощью симплекс-таблиц представлено на рисунках 1-3. Для способа решения с помощью симплекс-таблиц была написана программа.

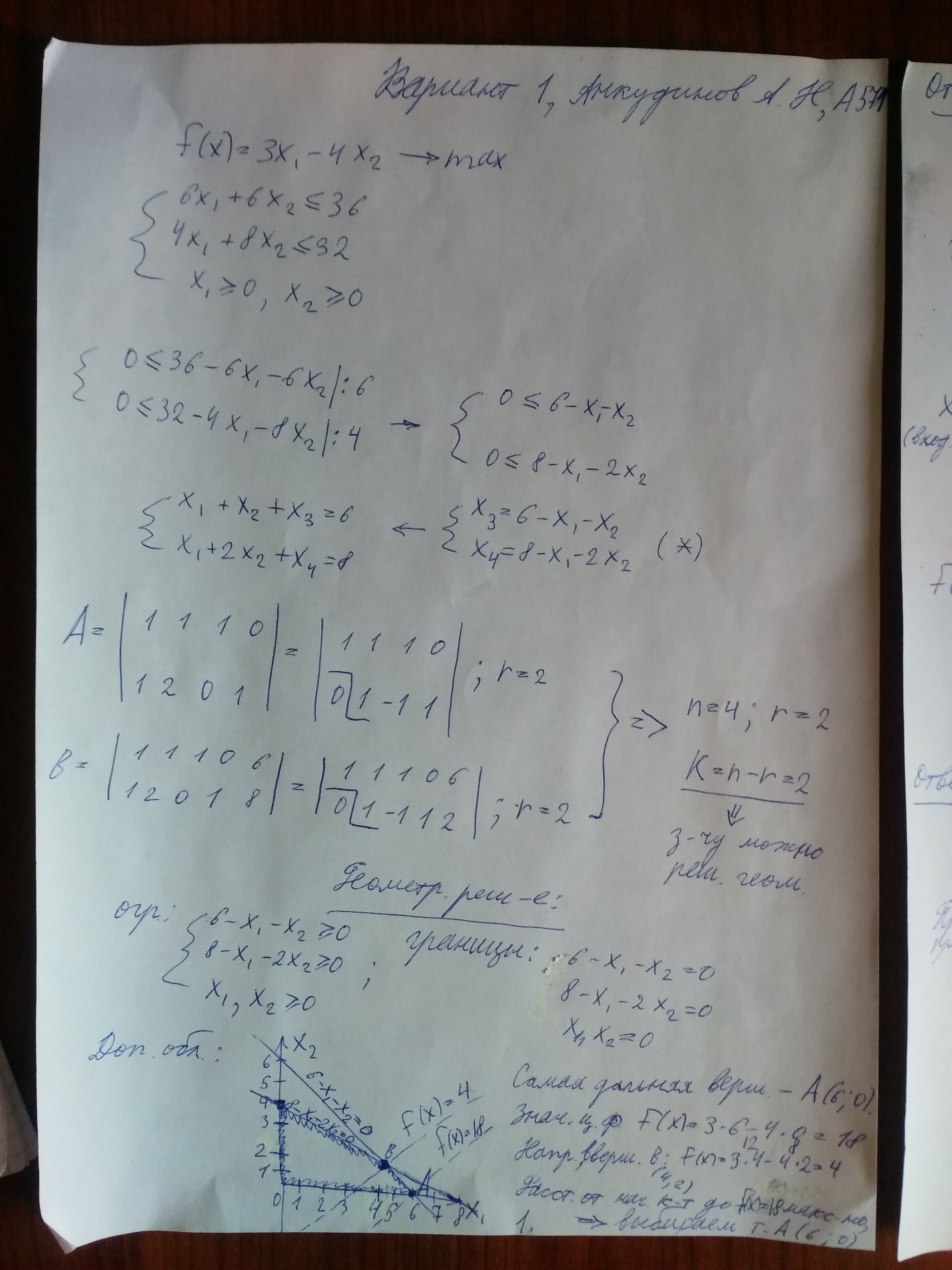
**

Рисунок 1

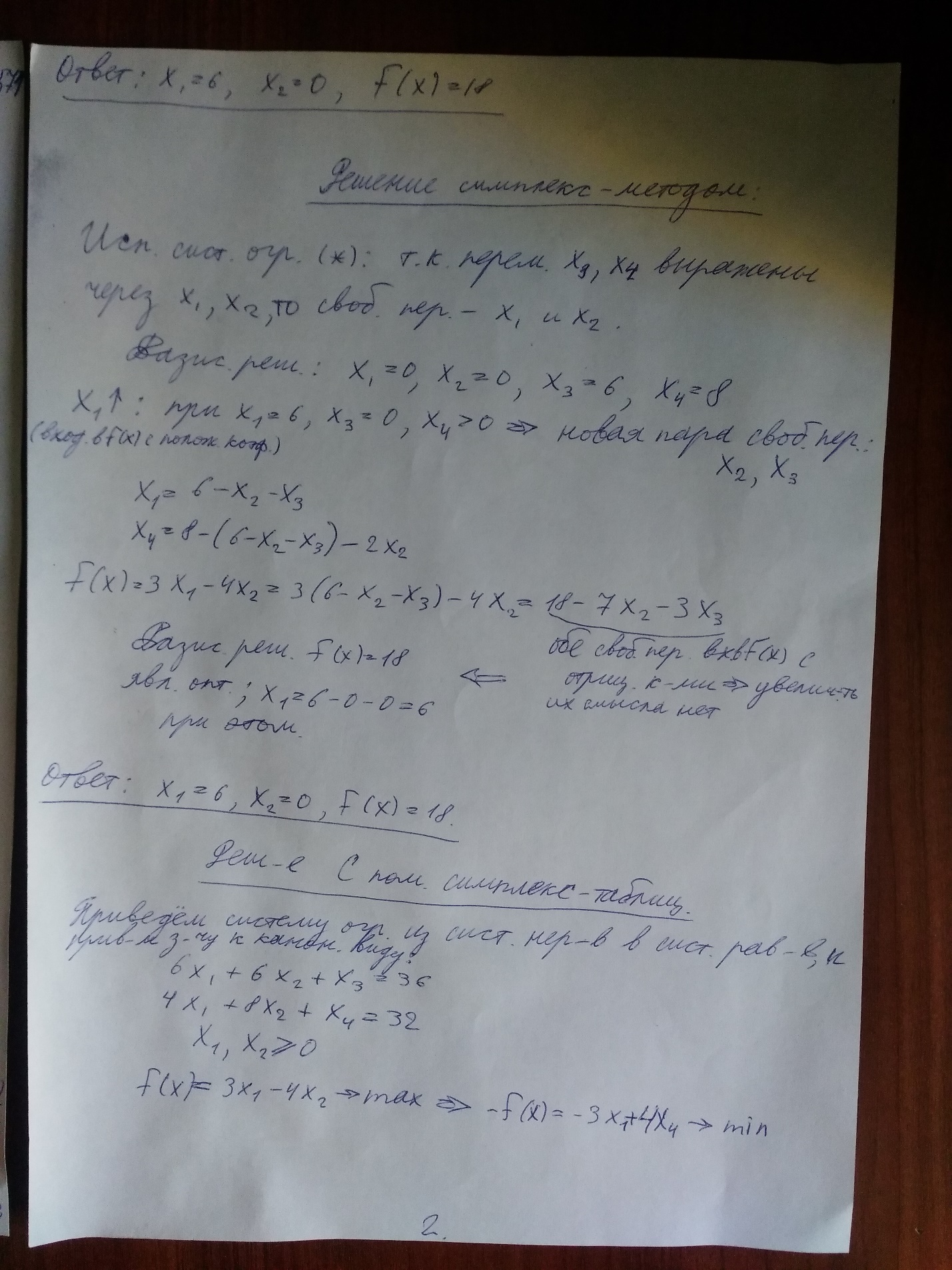


Рисунок 2

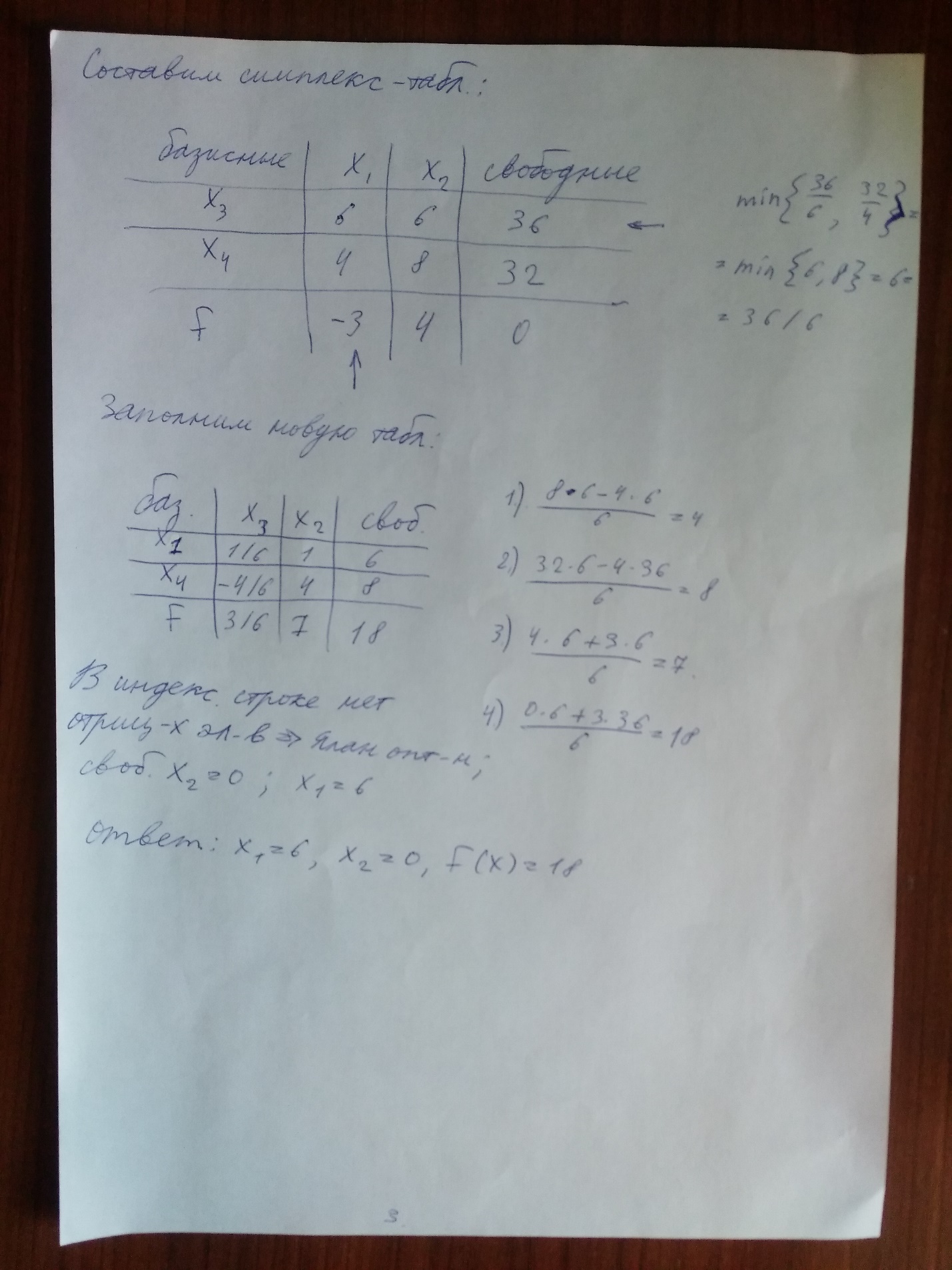


Рисунок 3

*Результат:*

|  |  |
| --- | --- |
|  | 6 |
|  | 0 |
|  | 18 |

*Вывод:* тремя вышеописанными способами было найдено решение задачи линейного программирования.

*Код программы*

#include <iostream>

#include <vector>

#include <string>

using namespace std;

const int Ni = 3, Nj = 3;

int col, row;

string titles[Ni+1][Nj+1] = // таблица заголовков

{

{"basis", "x1", "x2", "free"},

{"x3", "", "", ""},

{"x4", "", "", ""},

{"F", "", "", ""},

};

double table[Ni][Nj] = // таблица значений элементов

{

{6, 6, 36},

{4, 8, 32},

{-3, 4, 0}

};

double newtable[Ni][Nj]; // таблица значений элементов для пересчета на каждой итерации со значений старой таблицы

// прототипы функций

bool **there\_is\_a\_negativeElem\_in\_indexRow\_and\_there\_is\_a\_positiveElem\_in\_its\_column**();

void **find\_permissive\_column**();

void **find\_permissive\_row**();

void **changeVariables**();

void **fill\_newtable**();

void **print\_newtable**();

void **fill\_table\_for\_next\_step**();

bool **there\_is\_a\_negativeElem\_in\_indexRow**();

int **main**()

{

while(there\_is\_a\_negativeElem\_in\_indexRow\_and\_there\_is\_a\_positiveElem\_in\_its\_column()) // пока выполняется условие 3

{

find\_permissive\_column(); // ищем разрешающий столбец

find\_permissive\_row(); // ищем разрешающую строку

changeVariables(); // меняем переменные местами: одна выходит из базиса - другая входит в базис

fill\_newtable(); // рассчитываем значения в новой таблице

print\_newtable(); // вывод новой таблицы в консоль

fill\_table\_for\_next\_step(); // table = newtable - для следующей итерации

}

// вышли из цикла => условие 3 не выполнилось

if(there\_is\_a\_negativeElem\_in\_indexRow() == false) // проверка на условие 1 после проверки на условие 3 в цикле while

cout << "Plan is optimal, F = " << newtable[Ni-1][Nj-1]

<< ", x1 = " << newtable[row][Nj-1] << ", x2 = 0\n";

else // условия 1 и 3 не выполнились => условие 2

cout << "F decreases infinitely\n";

return 0;

}

bool **there\_is\_a\_negativeElem\_in\_indexRow\_and\_there\_is\_a\_positiveElem\_in\_its\_column**() // условие 3 - условие продолжения цикла

{

for(int j = 0; j < Nj; ++j)

if(table[Ni-1][j] < 0)

for(int i = 0; i < Ni-1; ++i)

if(table[i][j] > 0)

return true;

return false;

}

void **find\_permissive\_column**()

{

double min = table[Ni-1][0];

col = 0;

for(int j = 1; j < Nj; ++j)

if(table[Ni-1][j] < min)

{

min = table[Ni-1][j];

col = j;

}

}

void **find\_permissive\_row**()

{

double min;

if(table[0][col] > 0)

min = table[0][Nj-1] / table[0][col];

for(int i = 1; i < Ni-1; ++i)

if(table[i][col] > 0)

if(table[i][Nj-1] / table[i][col] < min)

{

min = table[i][Nj-1] / table[i][col];

row = i;

}

}

void **changeVariables**()

{

string temp;

temp = titles[row+1][0];

titles[row+1][0] = titles[0][col+1];

titles[0][col+1] = temp;

}

void **fill\_newtable**()

{

double elem = table[row][col];

newtable[row][col] = 1 / elem;

for(int j = 0; j < Nj; ++j) // заполнение разреш. строки

{

if(j == col) continue;

newtable[row][j] = table[row][j] / elem;

}

for(int i = 0; i < Ni; ++i) // заполнение разреш. столбца

{

if(i == row) continue;

newtable[i][col] = -table[i][col] / elem;

}

for(int i = 0; i < Ni; ++i) // заполнение остальных элементов таблицы

{

if(i == row) continue;

for(int j = 0; j < Nj; ++j)

{

if(j == col) continue;

newtable[i][j] = (table[i][j]\*elem - table[i][col]\*table[row][j]) / elem;

}

}

}

void **print\_newtable**()

{

cout << endl;

for(int j = 0; j < Nj+1; ++j)

cout << titles[0][j] << "\t";

cout << endl;

for(int i = 1; i < Ni+1; ++i)

{

cout << titles[i][0] << "\t";

for(int j = 0; j < Nj; ++j)

cout << newtable[i-1][j] << "\t";

cout << endl;

}

cout << endl;

}

void **fill\_table\_for\_next\_step**()

{

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

for(int j = 0; j < Nj; ++j)

table[i][j] = newtable[i][j];

}

bool **there\_is\_a\_negativeElem\_in\_indexRow**()

{

for(int j = 0; j < Nj; ++j)

if(table[Ni-1][j] < 0)

return true;

return false;

}