

Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

Методы оптимизации

Лабораторная работа №1 на тему:
«Постановка задачи линейного программирования»

Вариант 5

Преподаватель:

Коннова Н.С.

Студент:

Дудник А.И.

Группа:

ИУ8-34

Москва 2024

Цель работы

Изучить постановку задачи линейного программирования (ЛП); овладеть навыками решения задач ЛП с помощью симплекс-метода.

Постановка задачи

Требуется найти решение следующей задачи линейного программирования.

$$F=cx \rightarrow \max$$

$$Ax \leq b$$

$$x \geq 0$$

$$c:[3, 3, 7]$$

$$b:[3, 5, 7],$$

$$A:[[1, 1, 1]$$

$$[1, 4, 0]$$

$$[0, 0.5, 3]],$$

c - вектор коэффициентов целевой функции F ;

A - матрица системы ограничений;

b - вектор правой части системы ограничений.

Ход работы

Составим начальную симплекс-таблицу для дальнейших преобразований, добавив необходимые фиктивные переменные.

Базис	S	X1	X2	X3
X4	3	1	1	1
X5	5	1	4	0
X6	7	0	0.5	3
F	0	3	3	7

Опорный элемент находится в пересечении $(x_4, x_1) = 1$

Выполним перерасчет симплекс-таблицы, используя правило прямоугольника. Формула для обновления элемента будет следующей:

$$N_i = N(\text{старое значение}) - (A * B) / H,$$

$N(\text{старое значение})$ — это исходный элемент,

Н — разрешающий элемент,

А и В — старые элементы, которые формируют прямоугольник с элементами N(старое значение) и Н.

Базис	S	X4	X2	X3
X1	3	1	1	1
X5	2	-1	3	-1
X6	7	0	0.5	3
F	-9	-3	0	4

Проверим строку F на оптимальность

Пересечение $(F, x_3) > 0 \Rightarrow$ можно оптимизировать далее.

Опорный элемент находится в пересечении $(x_3, x_6) = 3$

Пересчитаем симплекс-таблицу

Базис	S	X4	X2	X6
X1	0.67	1	0.83	-0.33
X5	4.33	-1	3.17	0.33
X3	2.33	0	0.17	0.33
F	18.33	3	-0.67	-1.33

Критерий оптимальности по строке F пройден

Ответ: $x_1 = 0.67$, $x_2 = 0$, $x_3 = 2.33$. При этом максимизированная функция равна $F = 18.33$

Проверим решение подставив значения в исходную систему

1) $1 \cdot 0.67 + 1 \cdot 0 + 1 \cdot 2.33 \leq 3$ - выполняется

2) $1 \cdot 0.67 + 4 \cdot 0 + 0 \cdot 2.33 \leq 5$ - выполняется

3) $0 \cdot 0.67 + 0.5 \cdot 0 + 3 \cdot 2.33 \leq 7$ - выполняется

Вывод:

Сложность метода:

Время выполнения симплекс-метода может быть очень большим для некоторых задач.

Ограничения применимости:

Симплекс-метод работает только для задач линейного программирования, где целевая функция и ограничения являются линейными. Для нелинейных задач требуются другие методы оптимизации.

Преимущества симплекс-метода в наглядности и эффективности для большинства задач линейного программирования. Минусы же заключаются в том, что он не предназначен для задач нелинейного программирования.

Приложение А.

Файл *'simplex.cpp'*.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>
#include <cmath>
using namespace std;
class SMX_MD {
private:
    vector<double> c;
    vector<vector<double>> A;
    vector<double> b;
    vector<vector<double>> table;
    vector<string> FreeX;
    vector<string> DependX;
    string minormax;
    int coord_pred_razr_e_st = -1;
    int coord_pred_razr_e_str = -1;
public:
    SMX_MD(vector<double> c, vector<vector<double>> A,
vector<double> b, string minormax) {
        if (A.size() != b.size() || A[0].size() != c.size()) {
            throw runtime_error("Неверное соотношение размеров
матриц!");
        }
        if (minormax == "min") {
            this->c = c;
        }
        else {
            for (int i = 0; i < c.size(); i++) {
                c[i] = -c[i];
            }
            this->c = c;
        }
        this->A = A;
```

```

this->b = b;
this->minormax = minormax;
FillTable();
FreeX.resize(c.size());
int k = 0;
for (int i = 0; i < c.size(); i++) {
    FreeX[i] = "X" + to_string(i + 1);
    k++;
}
DependX.resize(b.size());
for (int i = 0; i < b.size(); i++) {
    k++;
    DependX[i] = "X" + to_string(k);
}
}
bool Solution() {
    cout << "Начальная задача: " << endl;
    Print();
    cout << endl;
    if (FindOptSolve()) {
        if (minormax == "max") {
            table[0][b.size()] = -table[0][b.size()];
            Print();
            cout << endl;
        }
        Check();
        return true;
    }
    return false;
}
bool FindOptSolve() {
    if (FindOprSolve()) {
        int l = 0;
        for (int t = 1; t < c.size() + 1; t++) if (table[t][b.size()] < 0) { l++; }
        if (l == c.size()) { return true; }
        for (int i = 1; i < c.size() + 1; i++) {

```

```

    if (table[i][b.size()] > 0) {
        int razr_stolb = i;
        double min = numeric_limits<double>::max();
        int flag2 = 0;
        int razr_str = -1;
        for (int z = 0; z < b.size(); z++) {
            double k = table[0][z] / table[razr_stolb][z];
            if (k < min && k > 0) {
                min = k;
                razr_str = z;
                flag2++;
            }
        }
        if (flag2 == 0) { continue; }
        if (razr_str != coord_pred_razr_e_str && razr_stolb !=
coord_pred_razr_e_st) {
            coord_pred_razr_e_st = razr_stolb;
            coord_pred_razr_e_str = razr_str;
            fix_table(razr_str, razr_stolb);
        }
        else {
            continue;
        }
        return FindOptSolve();
    }
}

return false;
}

void FillTable() {
    table.resize(c.size() + 1, vector<double>(b.size() + 1));

    for (int i = 0; i < b.size(); i++) {
        table[0][i] = b[i];
    }
}

```

```

table[0][b.size()] = 0;

for (int i = 0; i < b.size(); i++) {
    for (int j = 1; j < c.size() + 1; j++) {
        table[j][i] = A[i][j - 1];
    }
}

for (int i = 1; i < c.size() + 1; i++) { table[i][b.size()] = -c[i - 1]; }

bool FindOprSolve() {
    int l = 0;
    for (int t = 0; t < b.size(); t++) if (table[0][t] >= 0) { l++; }
    if (l == b.size()) { return true; }
    for (int i = 0; i < b.size(); i++) {
        if (table[0][i] < 0) {
            for (int j = 1; j < c.size() + 1; j++) {
                if (table[j][i] < 0) {
                    int razr_stolb = j;
                    double min = numeric_limits<double>::max();
                    int flag1 = 0;
                    int razr_str = -1;
                    for (int z = 0; z < b.size(); z++) {
                        double k = table[0][z] / table[razr_stolb][z];
                        if (k < min && k > 0) {
                            min = k;
                            razr_str = z;
                            flag1++;
                        }
                    }
                    if (flag1 == 0) { continue; }
                    if (razr_str != coord_pred_razr_e_str && razr_stolb !=
coord_pred_razr_e_st) {
                        coord_pred_razr_e_st = razr_stolb;
                        coord_pred_razr_e_str = razr_str;
                        fix_table(razr_str, razr_stolb);
                    }
                }
            }
        }
    }
}

```



```

        }
        else {
            continue;
        }
        return FindOprSolve();
    }
}
}
}
return false;
}
void fix_table(int razr_str, int razr_stolb) {
    vector<vector<double>> table1(c.size() + 1, vector<double>(b.size()
+ 1));
    double r_e = table[razr_stolb][razr_str];
    string _x = FreeX[razr_stolb - 1];
    FreeX[razr_stolb - 1] = DependX[razr_str];
    DependX[razr_str] = _x;
    table1[razr_stolb][razr_str] = 1 / r_e;
    for (int i = 0; i < c.size() + 1; i++) {
        if (i != razr_stolb) table1[i][razr_str] = table[i][razr_str] / r_e;
    }
    for (int i = 0; i < b.size() + 1; i++) {
        if (i != razr_str) table1[razr_stolb][i] = -table[razr_stolb][i] / r_e;
    }
    for (int i = 0; i < b.size() + 1; i++) {
        for (int j = 0; j < c.size() + 1; j++) {
            if (i != razr_str && j != razr_stolb) table1[j][i] = table[j][i] -
(table[razr_stolb][i] * table[j][razr_str]) / r_e;
        }
    }
    table = table1;
    Print();
    cout << endl;
}
void Print() {

```

```

cout << "\t";
cout << " S" << "\t";
for (const auto& obj : FreeX) { cout << " " << obj << "\t"; }
cout << endl;
cout << endl;
for (int i = 0; i < b.size() + 1; i++) {
    if (i != b.size()) cout << DependX[i] << "\t"; else cout << "F" << "\t";
    for (int j = 0; j < c.size() + 1; j++) {
        double value = round(table[j][i] * 100.0) / 100.0;
        if (value == 0) value = 0;
        if (value >= 0) cout << " " << value << "\t";
        else cout << value << "\t";
    }
    cout << endl;
}
cout << " _____" << endl;
}

void Check() {
    cout << "Ответ: " << endl;
    cout << endl;
    cout << "Функция: " << endl;
    vector<double> solve_x(table.size() + table[0].size() - 2);
    vector<string> str_X(solve_x.size());
    for (int i = 0; i < solve_x.size(); i++) str_X[i] = "X" + to_string(i + 1);
    for (int i = 0; i < DependX.size(); i++) {
        int k = find(str_X.begin(), str_X.end(), DependX[i]) - str_X.begin();
        solve_x[k] = table[0][i];
    }
    for (int i = 0; i < solve_x.size() / 2; i++) {
        if (minormax == "min" && i != solve_x.size() / 2 - 1) cout << c[i] <<
        "*" << solve_x[i] << " + ";
        if (minormax == "min" && i == solve_x.size() / 2 - 1) cout << c[i] <<
        "*" << solve_x[i] << " = " << table[0][table[0].size() - 1] << endl;
        if (minormax == "max" && i != solve_x.size() / 2 - 1) cout << -c[i]
        << "*" << solve_x[i] << " + ";
        if (minormax == "max" && i == solve_x.size() / 2 - 1) cout << -c[i]

```

```

<< "*" << solve_x[i] << " = " << table[0][table[0].size() - 1] << endl;
    }
    cout << endl;
    cout << "Проверка решения: " << endl;
    for (int i = 0; i < b.size(); i++) {
        for (int j = 0; j < solve_x.size() / 2; j++) {
            if (j != solve_x.size() / 2 - 1) cout << A[i][j] << "*" << solve_x[j] <<
" + ";
            if (j == solve_x.size() / 2 - 1) cout << A[i][j] << "*" << solve_x[j] <<
" <= " << b[i] << endl;
        }
    }
    cout << "Решения удовлетворяют ограничениям." << endl;
    cout << "_____ " << endl;
    cout << endl;
}
};
int main() {
    vector<double> c = { 3, 3, 7 };
    vector<double> b = { 3, 5, 7 };
    vector<vector<double>> A = {
        {1, 1, 1},
        {1, 4, 0},
        {0, 0.5, 3}
    };
    SMX_MD t(c, A, b, "max");
    if (!t.Solution()) {
        cout << "Решение не найдено" << endl;
    }
}

```

Terminal:

Начальная задача:

S	X1	X2	X3
---	----	----	----

X4	3	1	1	1
X5	5	1	4	0
X6	7	0	0.5	3
F	0	3	3	7

	S	X4	X2	X3
X1	3	1	1	1
X5	2	-1	3	-1
X6	7	0	0.5	3
F	-9	-3	0	4

	S	X4	X2	X6
X1	0.67	1	0.83	-0.33
X5	4.33	-1	3.17	0.33
X3	2.33	0	0.17	0.33
F	-18.33	-3	-0.67	-1.33

	S	X4	X2	X6
X1	0.67	1	0.83	-0.33
X5	4.33	-1	3.17	0.33
X3	2.33	0	0.17	0.33
F	18.33	-3	-0.67	-1.33

Ответ:

Функция:

$$3 \cdot 0.666667 + 3 \cdot 0 + 7 \cdot 2.33333 = 18.3333$$

Проверка решения:

$$1*0.666667 + 1*0 + 1*2.33333 \leq 3$$

$$1*0.666667 + 4*0 + 0*2.33333 \leq 5$$

$$0*0.666667 + 0.5*0 + 3*2.33333 \leq 7$$

Решения удовлетворяют ограничениям.
