Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Н.Э. БАУМАНА

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

МЕТОДЫ ОПТИМИЗАЦИИ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ В ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Лабораторная работа №2 на тему:

«Двойственность в линейном программировании»

Вариант 6

Преподаватель:
Коннова Н.С.

Студент:
Асатрян А.

Группа:
ИУ8-73

Москва 2019

Цель работы

Научиться по прямой задаче (ПЗ) ЛП формулировать и решать

соответствующую двойственную задачу (ДЗ).

Постановка задачи

Даны условия следующей задачи:

$$F = \boldsymbol{cx} \ \to max,$$
$$\boldsymbol{Ax} \leq \boldsymbol{b},$$
$$\boldsymbol{x} \geq 0.$$

Здесь $\boldsymbol{x} = [x_1, x_2, \dots, x_n]^T$ – искомый вектор решения;

$\boldsymbol{c} = [c_1, c_2, \dots, c_n]$ – вектор коэффициентов целевой функции (ЦФ) F;

$\boldsymbol{A} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$ – матрица системы ограничений;

$\boldsymbol{b} = [b_1, b_2, \dots, b_m]^T$ – вектор правой части системы ограничений.

Надо привести задачу к виду

$$\Phi = \boldsymbol{b^T y} \ \to min,$$
$$\boldsymbol{A^T y} \geq \boldsymbol{c^T},$$
$$\boldsymbol{y} \geq 0.$$

Здесь $\boldsymbol{y} = [y_1, y_2, \dots, y_n]^T$ – искомый вектор решения;

Ход работы.

Имеем следующие данные:

c =[2, 6, 7];

A = $\begin{bmatrix} 3 & 1 & 1 \\ 1 & 2 & 0 \\ 0 & 0.5 & 2 \end{bmatrix}$;

b =[3 8 1];

Приведем задачу ЛП к канонической форме, введя фиктивные переменные x4, x5, x6:

$$F = 2x_1 + 6x_2 + 7x_3 \rightarrow min$$

$$\begin{cases} 3x_1 + x_2 + x_3 + x_4 = 3; \\ x_1 + 2x_2 + x_5 = 8; \\ 0.5x_2 + 2x_3 + x_6 = 1; \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0.$$

x_4 , x_5, x_6 – базисные переменные;

x_1 , x_2, x_3 - свободные переменные.

Тогда имеем:

$$F = -(-2x_1 - 6x_2 - 7x_3) \rightarrow min$$

$$\begin{cases} x_4 = 3 - (3x_1 + x_2 + x_3) \\ x_5 = 8 - (x_1 + 2x_2) \\ x_6 = 1 - (0.5x_2 + 2x_3) \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0.$$

Исходная симплекс-таблица записывается в виде:

	S_0	x_1	x_2	x_3
x_4	3	3	1	1
x_5	8	1	2	0
x_6	1	0	0.5	2
F	0	2	6	7

Так как все элементы столбца S_0, кроме коэффициента целевой функции, неотрицательны, имеем опорное решение:

$x_1 = x_2 = x_3 = 0$, $x_4 = 3$, $x_5 = 8$, $x_6 = 1$.

Целевая функция $F = 0$.

Возьмем столбец с максимальным по модулю числом 7 и заменим переменные x_4 на x_1.

Получим преобразованную симплекс-таблицу:

	S_0	x_4	x_2	x_3
x_1	1	0.33	0.33	0.33
x_5	7	-0.33	1.67	-0.33
x_6	1	0	0.5	2
F	-2	-0.67	5.33	6.33

Анализируем последнюю строку симплекс-таблицы и ищем первый минимальный по модулю элемент: в столбце x_2 (разрешающий столбец)

Найдем минимальное положительное отношение элемента свободных членов S_0 к соответствующему элементу в разрешающем столбце:

$$\frac{1}{0.33} < \frac{1}{0.5}$$

Следовательно, x_2 –разрешающая строка.

Заменим базисную переменную x_6 на свободную x_2.

Получим преобразованную симплекс-таблицу:

	S_0	x_4	x_5	x_3
x_1	0.33	0.33	-0.67	-1
x_2	3.67	-0.33	-3.33	-7
x_6	2	0	2	4
F	-12.67	-0.67	-10.67	-15

Новое решение имеет вид:

$x_1 = 0.33$
$x_2 = 2$

$x_3 \ = 0$
$x_4 \ = 0$
$x_5 \ = 3.67$
$x_6 \ = 0$
Целевая функция $F = 12.67$

Подставим получившиеся значения в исходную задачу :

$F = 2 \cdot 0.33 + 6 \cdot 2 + 7 \cdot 0 = 12.67$

$$\begin{cases} 0.99 + 2 + 0 \leq 3; \\ 0.33 + 4 \leq 8; \\ 2 \cdot 0.5 + 0 \leq 1; \end{cases}$$

$x_1, x_2, x_3 \ \geq 0$

Верно!

Перейдем к двойственной задаче ЛП:

$$\Phi = \boldsymbol{b^T y} \ \rightarrow min,$$
$$\boldsymbol{A^T y} \geq \boldsymbol{c^T},$$
$$\boldsymbol{y} \geq 0.$$

c^T =[2 6 7]^T;

$A^T = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 0.5 \\ 1 & 0 & 2 \end{bmatrix}$;

b^T =[3 8 1];

$$\Phi = 3y_1 \ + 8y_2 \ + 1y_3 \ \rightarrow min$$

$$\begin{cases} 3y_1 + y_2 - y_4 = 2; \\ y_1 + 2y_2 + 0.5y_3 - y_5 = 6; \\ y_1 + 2y_3 \ - y_6 = 7; \end{cases}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \ \geq 0.$$

Тогда имеем:

$$\Phi = -(-3y_1 \ - 8y_2 \ - 1y_3\ ) \rightarrow min$$

$$\begin{cases} y_4 = -2-(-3y_1 - y_2); \\ y_5 = -6 - (-y_1 - 2y_2 - 0.5y_3); \\ y_6 = -7 - (-y_1 - 2y_3\ ); \end{cases}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \ \geq 0.$$

Исходная симплекс-таблица:

	S_0	y_1	y_2	y_3
y_4	-2	-3	-1	0
y_5	-6	-1	-2	-0.5
y_6	-7	-1	0	-2
Φ	0	-3	-8	-1

Ищем опорное решение:

В столбце S_0 отрицательный элемент в строке y_4. Ищем разрешающий столбец (первый отрицательный элемент в строке y_4) $-\ y_1$.

Заменим базисную переменную y_4 на свободную y_1.

Получим преобразованную симплекс-таблицу:

	S0	y_4	y_2	y_3
y_1	0.667	-0.333	0.333	0
y_5	-5.333	-0.333	-1.667	-0.5
y_6	-6.333	-0.333	0.333	-2
Ф	2	-1	-7	-1

В столбце S0 отрицательный элемент в строке y_5. Ищем разрешающий столбец (первый отрицательный элемент в строке y_5) $-\ y_4$.

Заменим базисную переменную y_5 на свободную y_4.

Получим преобразованную симплекс-таблицу:

	S0	y_5	y_2	y_3
y_1	6	-1	2	0.5
y_4	16	-3	5	1.5
y_6	-1	-1	2	-1.5
Ф	18	-3	-2	0.5

В столбце S0 отрицательный элемент в строке y_6. Ищем разрешающий столбец (первый отрицательный элемент в строке y_6) $-\ y_5$.

Заменим базисную переменную y_6 на свободную y_5.

Получим преобразованную симплекс-таблицу:

	S0	y_6	y_2	y_3
y_1	7	-1	0	2
y_4	19	-3	-1	6
y_5	1	-1	-2	1.5

Ф	21	-3	-8	5

Так как все элементы столбца S_0 неотрицательны, имеем опорное решение.

Проанализируем последнюю строку симплекс-таблицы и найдем первый положительный элемент: 5 в столбце y_3 (разрешающий столбец).

Найдем минимальное положительное отношение элемента свободных членов S_0 к соответствующему элементу в разрешающем столбце, следовательно, y_5 –разрешающая строка.

Заменим базисную переменную y_5 на свободную y_3.

Получим преобразованную симплекс-таблицу:

	S_0	y_6	y_2	y_5
y_1	5.667	0.333	2.667	-1.333
y_4	15	1	7	-4
y_3	0.667	-0.667	-1.333	0.667
Ф	17.667	0.333	-1.333	-3.333

Проанализируем последнюю строку симплекс-таблицы и найдем первый положительный элемент: 0.333 в столбце y_6 (разрешающий столбец).

Найдем минимальное положительное отношение элемента свободных членов S_0 к соответствующему элементу в разрешающем столбце, следовательно, y_4 – разрешающая строка.

Заменим базисную переменную y_6 на свободную y_4.

Получим преобразованную симплекс-таблицу:

	S_0	y_4	y_2	y_5
y_1	0.667	-0.333	0.333	0
y_6	15	1	7	-4

y_3	10.667	0.667	3.333	-2
Φ	12.667	-0.333	-3.667	-2

Новое решение имеет вид:

$y_1 = 0.667$
$y_2 = 0$
$y_3 = 10.667$
$y_4 = 0$
$y_5 = 0$
$y_6 = 15$
Целевая функция $\Phi = 12.667$

Так как в последней строке больше нет положительных элементов, имеем оптимальное решение:

Подставим получившиеся значения в исходную задачу :

$$\Phi = 3 \cdot 0.667 + 8 \cdot 0 + 1 \cdot 10.667 = 12.667$$

$$\begin{cases} 3 \cdot 0.667 + 0 + 0 \geq 1; \\ 0.667 + 0 + 0.5 \cdot 10.667 \geq 3; \\ 0.667 + 0 + 2 \cdot 10.667 \geq 8; \end{cases}$$

$y_1, y_2, y_3 \geq 0.$

Верно!

Целевые функции F и Ф равны. Выполняется принцип двойственности, значит ДЗ ЛП была составлена и решена правильно.

Вывод:

В ходе работы был изучен симплекс-метод решения задачи ЛП и двойсвенной задачи ЛП. Была решена конкретно поставленная задача и получено оптимальное значение целевой функции. Ответы, полученные при ручных вычислениях, совпали с ответами в программе и прошли проверку, поэтому решение верное.

Приложение

https://github.com/lightman1998/lab_02

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>

using namespace std;

class SimplexMethod {
	// строка таблицы
	struct Row {
		vector<double> a; // коэффициенты при x
		double b; // правая часть
	};

	int n; // число переменных
	int m; // число ограничений
	vector<Row> table; // симплекс таблица
	vector<double> c; // коэффициенты оптимизируемой функции
	vector<int> variables; // все переменные
	vector<int> basis; // базисные переменные
	vector<double> deltas; // дельты

	void CalculateDeltas(); // вычисление дельт

	int GetArgMinDelta(); // вычисление номера минимальной дельты
	int GetArgMaxDelta(); // вычисление номера максимальной дельты

	void InitialVariables(); // инициализация переменных и базиса

	void MakeNewBasis(double pivot, int index, int jindex); // задание
нового базисного элемента
	int MaxNegativeB(); // максимальная по модулю отрицательная b

public:
	SimplexMethod(int n, int m); // конструктор из размеров

	SimplexMethod(); // конструктор по умолчанию со всеми данными
	SimplexMethod(const SimplexMethod& method); // конструктор для
двойственной задачи из прямой

	void Read(); // ввод значений
	void Print(); // вывод таблицы

	void Solve(int max); // решение ОЗЛП
	void RemoveNegativeB(); // удаление отрицательных b
	double F; // результат оптимизации
};

SimplexMethod::SimplexMethod(int n, int m) {
	this->n = n; // запоминаем количество переменных
	this->m = m; // запоминаем количество условий

	table = vector<Row>(m, { vector<double>(n), 0 }); // создаём таблицу
	c = vector<double>(n); // создаём вектор коэффициентов
}


// конструктор по умолчанию со всеми данными
SimplexMethod::SimplexMethod() {
	// задаём значения для конкрутной задачи
```

```
        n = 3;
        m = 3;

        table = vector<Row>(m, { vector<double>(n), 0 });
        c = vector<double>(n);

        c = {2, 6, 7};

        table[0].a = {3, 1, 1};
        table[0].b = {3};

        table[1].a = {1, 2, 0};
        table[1].b = {8};

        table[2].a = {0, 0.5, 2};
        table[2].b = {1};

        InitialVariables(); // инициализируем переменные
}

// конструктор для двойственной задачи из прямой
SimplexMethod::SimplexMethod(const SimplexMethod& method) {
        // запоминаем размеры
        n = method.n;
        m = method.m;

        table = vector<Row>(m, { vector<double>(n), 0 }); // создаём таблицу
        c = vector<double>(n); // создаём вектор коэффициентов

        // функции цели присваиваем значения правой части прямой задачи
        for (int i = 0; i < n; i++)
                c[i] = method.table[i].b;

        // правой части присваиваем значения функции цели прямой задачи * -
1(так как сразу меняем знак для приведения к неравенствам вида <=)
        for (int i = 0; i < m; i++)
                table[i].b = method.c[i] * -1;

        // транспонируем матрицу прямой задачи и умножаем элементы на -1 (так
как сразу меняем знак для приведения к неравенствам вида <=)
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                        table[i].a[j] = method.table[j].a[i] * -1;
                }
        }

        InitialVariables(); // инициализируем переменные
}


// инициализация переменных и базиса
void SimplexMethod::InitialVariables() {
        variables.clear(); // очищаем переменные

        // добавляем переменные
        for (int i = 0; i < n; i++)
                variables.push_back(i);

        for (int i = 0; i < m; i++) {
                c.push_back(0); // добавляем нули в функцию
                variables.push_back(n + i); // добавляем доп переменные
                basis.push_back(n + i); // делаем их базисными
```

```
                // добавляем коэффициенты для переменных с коэффициентом 1,
если они стоят на главной диагонали, иначе с нулём
                for (int j = 0; j < m; j++)
                        table[i].a.push_back(i == j);
        }
}


// поиск макисмальной отрицательной b
int SimplexMethod::MaxNegativeB() {
        int imax = -1;

        // ищем максимальный отрицательный элемент
        for (int i = 1; i < n; i++) {
                if (table[i].b < 0 && (imax == -1 || table[i].b <
table[imax].b))
                        imax = i;
        }

        return imax; // возвращаем максимум
}

// устранение отрицательной правой части
void SimplexMethod::RemoveNegativeB() {
        int imax = MaxNegativeB(); // индекс максимального по модулю
отрицательного элемента

        // пока если отрицательные элементы
        while (imax != -1) {
                int jmax = 0; // индекс новой базисной переменной

                // идём по столбцу и ищем максимальный по модул. элемент
                for (int j = 1; j < m; j++) {
                        if (fabs(table[imax].a[j]) > fabs(table[imax].a[jmax]))
                                jmax = j;
                }

                basis[imax] = jmax;    // запоминаем индекс новой базисной
переменной
                MakeNewBasis(table[imax].a[jmax], imax, jmax); // делаем этот
элемент базисным

                imax = MaxNegativeB(); // находим новый максимальный по модулю
элемент в правой части
        }
}

// создание новой базисной переменной на месте index, jindex
void SimplexMethod::MakeNewBasis(double pivot, int index, int jindex) {
        // делим строку на элемент
        for (size_t i = 0; i < table[index].a.size(); i++)
                table[index].a[i] /= pivot;

        table[index].b /= pivot;

        // вычитаем из всех остальных строк эту строку, умноженную на элемент
в столбце jmax
        for (int i = 0; i < m; i++) {
                if (i == index)
                        continue;

                double value = table[i].a[jindex];

                for (size_t j = 0; j < table[i].a.size(); j++)
```

```
                    table[i].a[j] -= table[index].a[j] * value;

            table[i].b -= table[index].b * value;
        }
}


// ввод значений
void SimplexMethod::Read() {
        cout << "Enter function coefficients (c): ";
        c = vector<double>(n); // создаём вектор коэффициентов

        // считываем коэффициенты оптимизируемой функции
        for (int i = 0; i < n; i++)
                cin >> c[i];

        cout << "Enter restrictions coefficients:" << endl;

        // считываем коэффициенты ограничений
        for (int i = 0; i < m; i++) {
                cout << "Enter restriction " << (i + 1) << ": ";

                for (int j = 0; j < n; j++)
                        cin >> table[i].a[j];

                cin >> table[i].b;
        }

        variables.clear(); // очищаем переменные

        // добавляем переменные
        for (int i = 0; i < n; i++)
                variables.push_back(i);

        for (int i = 0; i < m; i++) {
                c.push_back(0); // добавляем нули в функцию
                variables.push_back(n + i); // добавляем доп переменные
                basis.push_back(n + i); // делаем их базисными

                // добавляем коэффициенты для переменных с коэффициентом 1,
если они стоят на главной диагонали, иначе с нулём
                for (int j = 0; j < m; j++)
                        table[i].a.push_back(i == j);
        }
}

// вывод таблицы
void SimplexMethod::Print() {
        int vars = variables.size();

        cout << endl;
        cout << "+-----+";

        for (int i = 0; i < vars; i++)
                cout << "-----------+";

        cout << endl;

        cout << "|  C  |";

        for (int i = 0; i < vars; i++)
                cout << " " << setw(9) << c[i] << " |";

        cout << endl;
```

```
	cout << "+-----+";

	for (int i = 0; i <= vars; i++)
		cout << "-----------+";

	cout << endl;

	cout << "|basis|";
	for (int i = 0; i < vars; i++)
		cout << "    x" << setw(2) << left << (i + 1) << "    |";

	cout << "     b     |" << endl;
	cout << "+-----+";

	for (int i = 0; i <= vars; i++)
		cout << "-----------+";

	cout << endl;

	for (int i = 0; i < m; i++) {
		cout << "| x" << setw(2) << left;

		if ((size_t)i < basis.size())
			cout << (basis[i] + 1);
		else
			cout << "?";

		cout  << " |";

		for (size_t j = 0; j < table[i].a.size(); j++)
			cout << " " << setw(9) << table[i].a[j] << " |";

		cout << " " << setw(9) << table[i].b << " |" << endl;
	}

	cout << "+-----+";

	for (int i = 0; i <= vars; i++)
		cout << "-----------+";

	cout << endl;

	if (!deltas.size())
		return;

	cout << "|  D  |";

	for (size_t i = 0; i < deltas.size(); i++)
		cout << " " << setw(9) << deltas[i] << " |";

	cout << endl;

	cout << "+-----+";

	for (int i = 0; i <= vars; i++)
		cout << "-----------+";

	cout << endl;
}

// вычисление дельт
void SimplexMethod::CalculateDeltas() {
	deltas.clear(); // очищаем массив дельт
```

```
        // проходимся по всем переменным
        for (size_t i = 0; i <= variables.size(); i++) {
                double delta = 0;

                // вычилсяем дельту
                for (size_t j = 0; j < basis.size(); j++)
                        delta += c[basis[j]] * (i < variables.size() ?
table[j].a[i] : table[j].b);

                // вычитаем коэффициент функции
                if (i < variables.size())
                        delta -= c[i];

                deltas.push_back(delta); // добавляем дельту в массив
        }
}

// вычисление номера минимальной дельты
int SimplexMethod::GetArgMaxDelta() {
        int imax = 0; // считаем, что первая дельта максимальна

        // проходимся по всем дельтам
        for (size_t i = 1; i < deltas.size() - 1; i++)
                if (deltas[i] > deltas[imax]) // если дельта стала больше
максимальной
                        imax = i; // обновляем индекс максимума

        return imax; // возвращаем индекс максимума
}

// вычисление номера минимальной дельты
int SimplexMethod::GetArgMinDelta() {
        int imin = 0; // считаем, что первая дельта минимальная

        // проходимся по всем дельтам
        for (size_t i = 1; i < deltas.size() - 1; i++)
                if (deltas[i] < deltas[imin]) // если дельта стала меньше
минимальной
                        imin = i; // обновляем индекс минимума

        return imin; // возвращаем индекс минимума
}

// решение ОЗЛП  max = 1 при минимизации max = -1 при максимизации
void SimplexMethod::Solve(int max) {
        int iteration = 1; // начинаем с первой итерации

        while (true) {
                CalculateDeltas(); // рассчитываем дельты
                int jmax;

                // если минимизируем
                if (max == 1)
                        jmax = GetArgMaxDelta(); // ищем индекс максимальной
                // если максимизация
                else
                        jmax = GetArgMinDelta(); // ищем индекс минимальной

                double maxDelta = deltas[jmax]; // получаем максимальную
дельту

                cout << (max == 1 ? "Min" : "Max") << " delta: " << maxDelta
<< endl; // выводим максимальную дельту
```

```
		// если она не положительна(или неотрицатльная для 
максимизации)
		if (maxDelta * max <= 0) {
			cout << "Plan is OK" << endl; // выводим, что план 
оптимален
			Print(); // выводим таблицу
			break; // и выходим
		}

		cout << "Iteration " << iteration++ << ":" << endl; // выводим 
номер итерации
		cout << "Calculating deltas:" << endl;
		Print(); // выводим таблицу

		vector<double> Q(m); // создаём симплекс отношения
		int imin = -1;

		// идём по ограничениям
		for (int i = 0; i < m; i++) {
			if (table[i].a[jmax] == 0) { // если коэффициент равен 
0
				Q[i] = 0; // то отношение равно нулю
			}
			else {
				Q[i] = table[i].b / table[i].a[jmax]; // 
вычисляем результат отношения

				// если оно отрицательно, то идём дальше
				if (Q[i] < 0)
					continue;

				// иначе обновляем минимальное симплекс 
отношение
				if (imin == -1 || Q[i] < Q[imin])
					imin = i;
			}
		}

		// вывод Q
		cout << "Q: ";

		for (int i = 0; i < m; i++)
			cout << Q[i] << " ";

		basis[imin] = jmax; // делаем переменную базисноц
		double pivot = table[imin].a[jmax]; // получаем опорный 
элемент

		cout << "Min Q: " << Q[imin] << endl; // выводим минимальное 
симплекс отношение
		cout << "x" << (jmax + 1) << " is new basis variable" << endl; 
// выводим новую базисную переменную
		cout << "Divide row " << (imin + 1) << " by " << pivot << 
endl; // делим строку на элемент

		MakeNewBasis(pivot, imin, jmax);
	}

	cout << (max == 1 ? "Fmin: " : "Fmax: ") << deltas[n + m] << endl; // 
выводим минимальное значение фукнции
	F = deltas[n + m]; // запоминаем результат
}
```

```
int main() {
        SimplexMethod method; // прямой метод
        SimplexMethod dualMethod(method); // двойственный метод

        cout << "Initial simplex table for main task: ";
        method.Print(); // вывод таблицы прямой задачи

        cout << endl << endl << "Initial dual simplex table: ";
        dualMethod.Print(); // вывод таблицы двойственной задачи

        cout << endl << "Solve main task: " << endl << endl;
        method.Solve(-1); // решение прямой задачи максимизации

        cout << endl << endl << "Dual simplex table after removing negative b: 
";
        dualMethod.RemoveNegativeB(); // вывод таблицы двойственной задачи
        dualMethod.Print(); // вывод таблицы двойственной задачи

        cout << endl << "Solve dual task: " << endl << endl;
        dualMethod.Solve(1); // решение задачи минимизации

        cout << endl << "Results: " << endl << "F(main) = " << method.F << 
endl << "F(dual) = " << dualMethod.F << endl; // вывод результатов
}
```