Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

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

ТЕОРИЯ ПРИНЯТИЯ РЕШЕНИЙ В УСЛОВИЯХ ИНФОРМАЦИОННЫХ КОНФЛИКТОВ

Лабораторная работа №2 на тему:
«Формулировка и решение двойственной ЗЛП»

Вариант 3

Преподаватель:
Коннова Н.С.

Студент:
Андреев Г.С.

Группа:
ИУ8-71

Москва 2021

Цель работы
Научиться по прямой задаче ЛП формулировать и решать соответствующую двойственную задачу.

Постановка задачи
Пусть исходная ПЗ ЛП имеет вид:

$$F = cx \rightarrow max\ ,$$

$$Ax \leq b\ ,$$

$$x \geq 0.$$

Здесь $x = [x_1, x_2, x_3]^T$ – вектор решения;

$c = [2, 8, 3]$ – вектор коэффициентов *целевой функции* (ЦФ) F;

$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 0 & 0.5 & 1 \end{pmatrix}$ – матрица системы ограничений;

$b = [4, 6, 2]$ – вектор правой части системы ограничений.

Требуется по ПЗ ЛП сформулировать ДЗ ЛП и решить ее симплекс-методом аналогично тому, как это сделано в лабораторной работе №1, представив все промежуточные преобразования симплекс-таблиц. Получив оптимальное решение, необходимо проверить его на согласованность с принципом двойственности и осуществить подстановку.

Решение исходной ПЗ ЛП симплекс-методом
Оптимальная симплекс-таблица ПЗ ЛП:

	S0	X4	X6	X3
X1	0.5	1.75	-0.25	-1
X5	3	0.5	0.5	0
X2	0.5	-0.25	-0.25	1
F	25.5	1.25	3.25	3

Результат:

$x_1 = 0.5$

$x_1 = 0.5$

$x_3 = 0$

$F(x_1, x_2, x_3) = 25.5$

Каноническая форма записи двойственной задачи ЛП

Используя фиктивные переменные x_4, x_5, x_6 приведем исходную задачу к каноническому виду

$c=[4,6,2]$ – вектор коэффициентов *целевой функции* (ЦФ) F;

$A=\begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 0.5 \\ 1 & 0 & 1 \end{pmatrix}$ – матрица системы ограничений;

$b=[2,8,3]$ – вектор правой части системы ограничений.

$$F=4x_1+6x_2+2x_3 \rightarrow min$$

$$\begin{cases} 2x_1+x_2 \geq 2 \\ x_1+2x_2+1/2x_3 \geq 8 \\ x_1+x_3 \geq 3 \end{cases}$$

$$y_i \geq 0$$

Исходная симплекс таблица

	S0	Y1	Y2	Y3
Y4	-2	-2	-1	0
Y5	-8	-1	-2	-0.5
Y6	-3	-1	0	-1
F	0	-4	-6	-2

Промежуточные симплекс таблицы

Найдем опорное решение:

1) Индекс разрешающего элемента (0, 2):

	S0	Y1	Y4	Y3
Y2	2	2	-1	0
Y5	-4	3	-2	-0.5
Y6	-3	-1	0	-1
F	12	8	-6	-2

2) Индекс разрешающего элемента (2, 3):

	S0	Y1	Y4	Y6
Y2	2	2	-1	0
Y5	-2.5	3.5	-2	-0.5
Y3	3	1	0	-1

F	18	10	-6	-2

3) Индекс разрешающего элемента (1, 3):

	S0	Y1	Y4	Y5
Y2	2	2	-1	0
Y6	5	-7	4	-2
Y3	8	-6	4	-2
F	28	-4	2	-4

Опорное решение найдено. Найдем оптимальное решение:

1) Индекс разрешающего элемента (1, 2):

	S0	Y5	Y4	Y6
Y2	3.25	0.25	0.25	-0.5
Y3	1.25	-1.75	0.25	-0.5
Y1	3	1	-1	0
F	25.5	-0.5	-0.5	-3

Так как в последней строке нет положительных коэффициентов, кроме свободного члена, то данное решение является оптимальным:

$$\left(\begin{array}{c} y_1 = 3 \\ y_2 = 3.25 \\ y_3 = 1.25 \end{array} \right.$$

$$F = 25.5$$

Проверка

Подставив полученные значения, можно убедиться в правильности решения

$F(x_1, x_2, x_3) = 4*3 + 6*3.25 + 2*1.25$ – проверка значения целевой функции

Вывод

В ходе работы была изучена постановка обратной задачи линейного программирования и решение её симплекс-методом. Основная процедура которого заключается в заменах переменных базиса, что сводится к перерасчету коэффициентов в симплекс-таблицах, таким образом данная процедура легко формализуется для вычисления данного метода с помощью ЭВМ.

Листинг программы

```python
import sys
import yaml
import pydantic
import typing as t


class SimplexProblem(pydantic.BaseModel):
    c: t.List[float]
    A: t.List[t.List[float]]
    b: t.List[float]

    @classmethod
    def from_yaml(cls, filename: str) -> "SimplexProblem":
        with open(filename, "r") as f:
            data = yaml.load(f, yaml.CLoader)
            p = cls(**data)
            return p


class MinTowarding:
    def __str__(self):
        return 'min'


class MaxTowarding:
    def __str__(self):
        return 'max'


def print_separator():
    """
    Печать разделительной черты. Черта предназначена для того, чтобы
    разделять логические части решения задачи симплекс-методом
    """
    width = 40
    print()
    print('-' * width)


def print_system(c, A, b, towards_to, title):
    """
    Печать задачи линейного программирования.
    :param c: коэффициенты при линейной комбинации ЦФ
    :param A: матрица ограничений
    :param b: вектор ограничений
    :param towards_to: то, к чему мы стремим значение ЦФ
    :param title: информационное сообщение
    """

    def format_value(value, var_pos):
        """
        Форматирование коэффициента систем
        :param value: значение коэффициента
        :param var_pos: порядковый номер переменной, которой пренадлежит этот
коэффициент
        """
        if value > 0:
```

```python
            return '+{:4} x_{}'.format(value, var_pos + 1)
        elif value < 0:
            return '{:5} x_{}'.format(value, var_pos + 1)
        else:
            return '{:9}'.format('')

    print_separator()
    print(title)

    # печать целевой функции
    print('Целевая функция:')
    F = []
    for i in range(0, len(c)):
        F.append(format_value(c[i], i))
    print('F = {} --> {}'.format(' '.join(F), towards_to))

    # печать системы ограничений
    print('Ограничения:')
    for i in range(0, len(A)):
        row = []
        for j in range(0, len(A[i])):
            row.append(format_value(A[i][j], j))
        print('{{ {} = {}'.format(' '.join(row), b[i]))
    print('{{ x_i >= 0, i = 1, ... , {}'.format(len(A[0])))



def to_canonical(c, A, b):
    """
    :param c: коэффициенты при линейной комбинации ЦФ
    :param A: матрица ограничений
    :param b: вектор ограничений
    """
    # вычисляем количество фиктивных переменных
    fictious_vars = len(A)

    # переводим коэффициенты при линейной комбинации ЦФ в канонический вид
    # также добавляем нулевые коэффициенты при фиктивных переменных
    canonical_c = []
    for ci in c:
        canonical_c.append(-ci)
    for _ in range(0, fictious_vars):
        canonical_c.append(0)

    # переводим матрицу ограничений в канонический вид
    canonical_A = []
    for i in range(0, len(A)):
        # формирование строк
        canonical_A.append([])

        # копирование коффициентов для реальных переменных
        for j in range(0, len(A[i])):
            canonical_A[i].append(A[i][j])
        # создание коффициентов для фиктивных переменных
        for k in range(0, fictious_vars):
            if k == i:
                canonical_A[i].append(1)
            else:
                canonical_A[i].append(0)

    return canonical_c, canonical_A, b
```

```python
class Legend:
    def __init__(self, index):
        self._index = index

    def __str__(self):
        return 'x_{}'.format(self.index())

    def index(self):
        return self._index


class SimplexTable:
    """
    :param c: коэффициенты при линейной комбинации ЦФ
    :param A: матрица ограничений
    :param b: вектор ограничений
    """

    def __init__(self, c, A, b) -> None:
        # создание симплекс-таблицы
        table = []
        # заполнение ограничений
        for i in range(0, len(A)):
            table.append([])
            # свободные члены
            table[i].append(b[i])
            # коэффициенты при переменных
            for j in range(0, len(A[i]) - len(b)):
                table[i].append(A[i][j])
        # заполнение коэффициентов ЦФ
        table.append([0])
        for j in range(0, len(c) - len(b)):
            table[-1].append(-c[j])

        # создание верикальных и горизонтальных легенд
        # (легенд строк и столбцов)
        vert = []
        hor = ['Si']
        for i in range(0, len(A[0])):
            if i < len(b):
                hor.append(Legend(i + 1))
            else:
                vert.append(Legend(i + 1))
        vert.append('F')

        self._table = table
        self._vert_legends = vert
        self._hor_legends = hor

    def _exchange_basic_variables(self, r, k):
        """
        :param r: индекс разрешающей строки
        :param k: индекс разрешающего столбца
        """
        # обмен легенд разрешающих строки и столбца
        self._vert_legends[r], self._hor_legends[k] = self._hor_legends[k], self._vert_legends[r]

        # создание новой таблицы
```

```
        new_table = [[0 for _ in range(0, len(self._table[i]))] for i in range(0, len(self._table))]

        # применение правил создания новой таблицы
        new_table[r][k] = 1 / self._table[r][k]
        for j in range(0, len(self._table[r])):
            if j != k:
                new_table[r][j] = self._table[r][j] / self._table[r][k]
        for i in range(0, len(self._table)):
            if i != r:
                new_table[i][k] = - self._table[i][k] / self._table[r][k]
        for i in range(0, len(self._table)):
            for j in range(0, len(self._table[i])):
                if i != r and j != k:
                    new_table[i][j] = self._table[i][j] - self._table[i][k] * self._table[r][j] / self._table[r][k]

        self._table = new_table

    def print(self, titles):
        """
        Печать симплекс-таблицы
        :param titles: информационные сообщения
        """
        # создадим сетку, которую заполним элементами симплекс-таблицы
        grid = []

        # добавление горизонтальных легенд
        grid.append([''])
        for legend in self._hor_legends:
            grid[0].append(legend)

        # добавление вертикальных легенд и элеметов симплекс-таблицы
        for i in range(0, len(self._table)):
            grid.append([self._vert_legends[i]])
            for j in range(0, len(self._table[i])):
                grid[-1].append(round(self._table[i][j], 4))

        def print_grid_row(row):
            """
            Печать строки сетки
            :param row: строка, которую необходимо напечатать
            """
            cell_wigth = 8  # ширина клетки сетки
            cell_format = '{:' + str(cell_wigth) + '}'
            print('|'.join([cell_format.format(str(cell)) for cell in row]))

        print_separator()
        for title in titles:
            print(title)
        for row in grid:
            print_grid_row(row)

        # нахождение значений исходных переменных
        solution = self.solution()

        print('Значение исходных переменных для данной симплекс-таблицы:')
        for var in solution:
            print('x_{} = {:.4f}'.format(var[0], var[1]))
        print('Значение целевой функции при данных значениях исходных переменных:')
        print('F = {:.4f}'.format(self.calculate_target_function()))
```

```python
def is_pivot_solution(self):
    """
    Проверка на то, является ли данное состояние таблицы опорным
    """
    for i in range(0, len(self._table) - 1):
        if self._table[i][0] < 0:
            return False
    return True

def to_pivot_solution(self):
    """
    Преведение состояния таблицы к опорному решению
    """

    def find_resolving_elem(t):
        """
        Функция поиска разрешающего элемента
        """
        resolving_column = None
        # нахождение разрешающей строки путем анализа свободных членов и
        # коэффициентов при свободных переменных
        for i in range(0, len(t) - 1):
            if t[i][0] < 0:
                # если свободный член i отрицательный, то
                # ищем отрицательный элемент ij
                for j in range(0, len(t[i])):
                    if t[i][j] < 0:
                        resolving_column = j
                # если отрицательный элемент ij не найден,
                # то заключаем, что система несовместна
                if resolving_column == None:
                    raise ValueError('Система несовместна!')
                break

        # ищем разрешающую строку, находя минимальной частное si0/sik
        min_division = None
        resolving_row = None
        for i in range(0, len(t) - 1):
            if t[i][resolving_column] != 0:
                division = t[i][0] / t[i][resolving_column]
                if division > 0 and (min_division == None or division < min_division):
                    min_division = division
                    resolving_row = i

        return (resolving_row, resolving_column)

    # производим проверку на то, является ли решение опорным,
    # и возвращаемся из функции
    # если оно не опроное, то заменяем одну базисную переменную
    # с печатью промежуточного результата
    step = 0
    while True:
        step += 1
        if self.is_pivot_solution():
            return

        ij = find_resolving_elem(self._table)
        self._exchange_basic_variables(*ij)
        self.print([
            'Разрешающий элемент: {}'.format(ij),
            'Промежуточная симплекс-таблица №{}'.format(step)
```

```
        ])

    def is_optimal_solution(self):
        """
        Функция проверки решения для данной симплекс-таблицы на оптимальность
        """
        for j in range(1, len(self._table[-1])):
            if self._table[-1][j] > 0:
                return False
        return True

    def calculate_target_function(self):
        """
        Функция расчета значения ЦФ для данной сиплекс-таблицы
        """
        return self._table[-1][0]

    def to_optimal_solution(self):
        """
        Преведение состояния таблицы к опорному решению
        """

        def find_resolving_elem(t):
            """
            Функция поиска разрешающего элемента
            """
            # нахождение разрешающей строки путем анализа коэффициентов при ЦФ
            # разрешающий столбец располагается там, где коэффициент при свободной
            # переменной положителен (что говорит о неоптимальности данного решения)
            resolving_column = None
            for j in range(1, len(t[-1])):
                value = t[-1][j]
                if value > 0:
                    resolving_column = j
                    break

            # нахождение минимального положительного частного вида si0/sik
            min_division = None
            resolving_row = None
            for i in range(0, len(t) - 1):
                if t[i][resolving_column] != 0:
                    division = t[i][0] / t[i][resolving_column]
                    if division > 0 and (min_division == None or division < min_division):
                        min_division = division
                        resolving_row = i

            return (resolving_row, resolving_column)

        # производим проверку на то, является ли решение оптимальным,
        # и возвращаемся из функции
        # если оно не оптимально, то заменяем одну базисную переменную
        # с печатью промежуточного результата
        step = 0
        while True:
            step += 1
            if self.is_optimal_solution():
                return

            ij = find_resolving_elem(self._table)
            self._exchange_basic_variables(*ij)
            self.print([
```

```
            'Разрешающий элемент: {}'.format(ij),
            'Промежуточная симплекс-таблица №{}:'.format(step)
        ])

    def solution(self):
        """
        Получить базисное решение исходной задачи
        """
        # выделеим все переменные и отберем исходные
        solution = []
        for i in range(1, len(self._hor_legends)):
            solution.append((self._hor_legends[i].index(), 0))

        for j in range(0, len(self._vert_legends) - 1):
            solution.append((self._vert_legends[j].index(), self._table[j][0]))
        solution = sorted(solution, key=lambda var: var[0])

        return solution[:len(self._table) - 1]


def do_simplex_method(c, A, b):
    """
    :param c: коэффициенты при линейной комбинации ЦФ
    :param A: матрица ограничений
    :param b: вектор ограничений
    """
    print_system(c, A, b, MaxTowarding(), 'Исходная задача:')

    c, A, b = to_canonical(c, A, b)
    print_system(c, A, b, MinTowarding(), 'Канонический вид:')

    table = SimplexTable(c, A, b)
    table.print(['Исходная симплекс-таблица:'])

    print_separator()
    if table.is_pivot_solution():
        print('Исходная симплекс-таблица описывает опорное решение.')
        print('По этой причине незамедлительно переходим к следующему этапу симплекс-
метода.')
    else:
        print('Найдем опорное решение:')
        table.to_pivot_solution()
        table.print(['Найдено опорное решение:'])

    print_separator()
    print('Найдем оптимальное решение:')
    table.to_optimal_solution()
    table.print(['Результирующая симплекс-таблица:'])

    solution = table.solution()

    print_separator()
    print('Произведем проверку на соответствие для полученого решения и найденого
оптимального значения ЦФ:')
    print('F = ' + ' + '.join(
        ['{} * ({})'.format('x_{}'.format(prod[0][0]), round(prod[1], 4)) for prod in zip(solution, c)])
+ ' = ')
    print('  = ' + ' + '.join(
        ['{} * ({})'.format(round(prod[0][1], 4), round(prod[1], 4)) for prod in zip(solution, c)]) + '
= ')
```

```
    print('  = {:.4f}'.format(sum([prod[0][1] * prod[1] for prod in zip(solution, c)])))


if __name__ == '__main__':
    problem = SimplexProblem.from_yaml(sys.argv[1])

    do_simplex_method(problem.c, problem.A, problem.b)
```