МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Информационных систем

КУРСОВАЯ РАБОТА

по дисциплине «Теория принятия решений»

Тема: Применение методов линейного и динамического программирования для решения практических задач (по вариантам)

Вариант: 4 (330)

Студент гр. 6373 _____ Васильев М.С.

Преподаватель _____ Пономарев А.В.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Васильев М.С.

Группа 6373

Тема работы: Применение методов линейного и динамического программирования для решения практических задач (по вариантам)

Исходные данные:

Текст индивидуального задания на курсовую работу в соответствии с назначенным вариантом (см. https://avponomarev.bitbucket.io/tasks/4.pdf).

Содержание пояснительной записки: «Содержание», «Введение», «Задача инвестирования», «Задача распределения ресурсов», «Транспортная задача линейного программирования», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 21.02.2019

Дата сдачи курсовой работы: 17.04.2019

Дата защиты курсовой работы: 17.04.2019

Студент _____ Васильев М.С.

Преподаватель _____ Пономарев А.В.

АННОТАЦИЯ

В курсовой работе рассмотрено применение методов динамического и линейного программирования для решения практических задач. В процессе выполнения работы были использованы популярные библиотеки для решения выпуклых оптимизационных задач, такие как CVXOPT. Для решения динамического программирования был использован Python C API для более создания более быстрой версии решателя динамических оптимизационных задач. В результате работы получены оптимальные планы для решаемых задач и представлена естественная интерпретация полученных результатов.

SUMMARY

In the given course work we reviewed methods of dynamic and linear programming for solving practical problems. In progress of work we used libraries for solving convex optimization problems like CVXOPT. For solving dynamic problems, we used self-written library with Python C API, that increased the speed for solving dynamic problem. As a result, we've got optimal solutions for given tasks and presented the natural interpretation of them.

Оглавление

ВВЕДЕНИЕ

Целью данной работы было практическое решение линейных и динамических оптимизационных задач. В процессе выполнения работы были определены следующие задачи, такие как нахождение и оптимизация задачи динамического программирования, анализ чувствительности и его интерпретация в задачах линейного программирования. Для были использованы следующие библиотеки: для решения выпуклых оптимизационных задач - CVXOPT, для написания быстрого динамического решателя – Python C API, для визуализации и интерпретации полученных данных – Numpy, Pandas, Seaborn.

1. ЗАДАЧА ИНВЕСТИРОВАНИЯ

1.1. Условие задачи

В цехах N_1 и N_2 предприятия производится продукт Y, который в дальнейшем используется в качестве исходного материала для производства изделий в цехе N_3. Суммарная производительность цехов N_1 и N_2 зависит от вложения дополнительных средств X. При работе цехов N_1 и N_2 в течение одного месяца эта зависимость может быть приближено представлена в виде функций:

- $N_1: y = 5 + (x + 40)^{\frac{1}{3}}$
- $N_2: y = 7 + (x + 30)^{\frac{1}{3}}$

Функции остатка средств в течение месяца:

- $N_1: 0.8x$
- $N_2: 0.62x$

Средства, выделяемые на оба цеха в течение квартала (3 месяца), составляют 179 единиц; перераспределение производится помесячно.

Требуется распределить средства на планируемый квартал с целью получения максимального количества продукта Y.

1.2. Формализация задачи

Введём следующие обозначения:

- x_{ij} – инвестиции в i-й цех в j-м месяце ($i = 1 \dots 2$, $j = 1 \dots 3$)

Для введённых обозначений составим целевую функцию и определимся с направлением оптимизации, в нашем случае это максимум:

$$f(\vec{x}) = 24 + \sum_{k=1}^{3}[(x_{1k} + 40)^{\frac{1}{3}} + (x_{2k} + 30)^{\frac{1}{3}}] \to max$$

Запишем ограничения, связанные с функцией остатка средств в течение месяца:

$$0.8x_{1j} - \ x_{1(j+1)} = 0$$

$$0.62x_{2j} - x_{2(j+1)} = 0$$

$$j = 1..2$$

Запишем ограничения, связанные с количеством изначально выделенных средств:

$$\sum_{j=1}^{3}\sum_{i=1}^{2} x_{ij} \leq 179$$

Заметим, что все переменные неотрицательны, так как это противоречит условию задачи. Следовательно, на данном этапе формализации мы имеем задачу на выпуклом многогранном множестве с нелинейной целевой функцией.

Так как линейное программирование нам не подходит, воспользуемся динамическим программированием. Выберем способ описания процесса []:

- Этапы – месяц финансирования, может быть в интервале от 0 до 179
- Выигрыш – количество продукта Y
- Управление – количество финансов, выделяемое на первый цех (на второй цех получаем автоматически, вычитая количество из состояния), может быть в интервале от 0 до 179.
- Состояние – остаточное количество финансов на начало месяца, изначально 179.

Запишем выигрыш на i-м шаге:

$$w_i(S_i, u_i) = 12 + (u_i + 40)^{\frac{1}{3}} + (S_i - u_i + 30)^{1/3}$$

Запишем изменение состояния на i-м шаге:

$$\varphi_i(S_i, u_i) = 0.8u_i + 0.62\ (S_i - u_i) = 0.62S_i - 0.2u_i$$

Запишем основное функциональное уравнение

$$W_i(S_i) = \max_{u_i \in [0;179]} \{12 + (u_i + 40)^{\frac{1}{3}} + (S_i - u_i + 30)^{\frac{1}{3}} + W_{i+1}(0.62S_i - 0.2u_i)\}$$

1.3. Решение задачи

Для реализации задачи методом динамического программирования был выбран рекурсивный подход с мемоизацией, потому что он имеет лучшую читаемость, по сравнению с итеративным подходом, а разница по времени исполнения мала [].

Так как динамическое программирование подразумевает под собой рекурсивное нахождение большого количества значений функции, а в языке Python существуют большие накладные расходы на их вызов [], было принято решение написать расширение для Python [] на C++.

Вычисление производится при помощи рекурсивного метода прямой прогонки. Условием выхода из рекурсии является выход за допустимый предел этапов:

```
if (stage >= maxStage) {
    return std::make_pair(0, 0);
}
```

Поиск максимума производится прямым методом в цикле:

```
for (int managementStep = 0; managementStep * precision <= state; managementStep++) {
            stateDiff = local_state_change(state, managementStep * precision);
```

```
            profit = local_win(state, managementStep * precision) +
global_profit(stage + 1, stateDiff).first;

            if (profit > maxProfit) {
                maxProfit = profit;
                maxManagement = managementStep * precision;
            }
}
```

Мемоизация производится при помощи структуры cache класса DynamicSolver. Данная структура представляет из себя динамический массив неупорядоченных хэш-таблиц, где каждая хэш-таблица представляет собой историю вызовов функции на i-м этапе, где i – индекс в массиве (этап в терминах динамического программирования). При каждом вызове сохраняется локальное управление и локальный выигрыш.

Итоговый оптимальный план восстанавливается из структуры cache, при помощи последовательного нахождения изменения состояний при помощи сохраненных условно оптимальных управлений следующим образом:

```
std::vector<double> result;

        for (int stage = 0; stage < maxStage; ++stage) {
            result.push_back(cache[stage][state].second);
            state = local_state_change(state, result.back());
        }

return result;
```

В результате поиска решения получаем следующие данные (приведены с точностью до 0.1), которые иллюстрирует Рисунок 1.

```
Maximum y product is 64.02966582649583
Plan for this count of product is: [111.10000000000001, 73.9, 44.300000000000004]
```

Рисунок 1 – вывод решения задачи динамического программирования

2. ЗАДАЧА РАСПРЕДЕЛЕНИЯ РЕСУРСОВ

2.1. Условие задачи

Цех N_3 выпускает продукцию в виде трех изделий A, B и C в одинаковом количестве. Для изготовления каждого из видов изделий A, B и C в цехе N_3 может быть использована та или иная группа технологического оборудования. Расход продукта Y при изготовлении одного изделия указан в Таблица 1. В Таблица 2 приведены данные о фонде рабочего времени оборудования (в часах) и о времени, необходимом для изготовления одного изделия (в минутах).

Таблица 1 – Расход продукта Y при изготовлении одного изделия

	A	B	C
1	-	0.005	0.004
2	0.003	0.009	-
3	0.003	-	0.005

Таблица 2 – Временные параметры (в часах)

	A	B	C	Фонд рабочего времени
1	-	5	8	860
2	20	8	-	1500
3	13	-	9	870

Требуется спланировать работу оборудования цеха N3 в течение одного квартала с целью получения максимального количества изделий видов A, B, C; полученное решение необходимо исследовать:

1. Выяснить наличие в решении полностью загруженной группы оборудования;

2. Если такая группа оборудования в решении присутствует, средствами параметрического изменения правых частей исследовать влияние величины фонда рабочего времени этой группы оборудования на структуру решения (изменение фонда рабочего времени в сторону увеличения и уменьшения);
3. Если такая группа оборудования в решении отсутствует, средствами параметрического изменения правых частей предварительно увеличить количество используемого продукта Y до ее появления, а затем вернуться к п. 2.

2.2. Формализация задачи

Введём следующие обозначения:

- x_{ij}- количество продукта i, произведённое цехом j.
- $prev$ – количество продукции, полученное из решения первой задачи.

Составим целевую функцию по условию в соответствии с введёнными обозначениями, которая получается непосредственно из условия задачи:

$$f(\vec{x}) = \sum_{j=1}^{3}\sum_{i=1}^{3} x_{ij} \to max$$

Составим ограничения по количеству продукции, которые получаются путём непосредственной интерпретации таблицы 1 в введённых терминах:

$$0.005x_{21} + 0.004x_{31} + 0.003x_{12} + 0.009x_{22} + 0.003x_{13} + 0.005x_{33} \leq prev$$

Составим ограничения по фонду допустимого времени, которые получаются путём непосредственной интерпретации таблицы 2 в введённых терминах:

$$5x_{21} + 8x_{31} \leq 860$$ – 1-я строка таблицы 2

$$20x_{21} + 8x_{22} \leq 1500$$ – 2-я строка таблицы 2

$$13x_{13} + 9x_{33} \leq 870$$ – 3-я строка таблицы 2

Составим ограничения исходя из условия равенства количества производимой продукции всеми группами оборудования:

$$0.003x_{12} + 0.003x_{13} - 0.005x_{21} - 0.009x_{22} = 0$$ – количество продукции первого цеха равно количеству продукции второго цеха

$$0.003x_{12} + 0.003x_{13} - 0.004x_{31} - 0.005x_{33} = 0$$ – количество продукции первого цеха равно количеству продукции третьего цеха

Также, учитывая то, что количество произведённой продукции не может быть отрицательным, введём ограничения отрицательности переменных:

$$x_{ij} \geq 0, i = 1..3, j = 1..3$$

Все ограничения, а также целевая функция - линейны, все переменные – неотрицательны, следовательно перед нами задача линейного программирования.

2.3. Решение задачи

Для решения задачи линейного программирования будем использовать Python и библиотеку CVXOPT []. Запишем формализованные ограничения в матричной форме, неиспользуемые переменные опустим.

Соответственно матрица представления целевой функции выглядит, как:

`c = matrix([-1 for _ in range(6)], tc='d')`, что соответсвует столбцу состоящему из -1.

Матрица левых частей ограничений неравенств:

```
G = matrix([[0.005, 0.004, 0.003, 0.009, 0.003, 0.005],
            [5, 8, 0, 0, 0, 0],
            [0, 0, 20, 8, 0, 0],
            [0, 0, 0, 0, 13, 9],
```

```
            [-1, 0, 0, 0, 0, 0],
            [0, -1, 0, 0, 0, 0],
            [0, 0, -1, 0, 0, 0],
            [0, 0, 0, -1, 0, 0],
            [0, 0, 0, 0, -1, 0],
            [0, 0, 0, 0, 0, -1]], tc='d')
```

Матрица правых частей ограничений неравенств:

`h = matrix([max_y_product, 860, 1500, 870, 0, 0, 0, 0, 0, 0], tc='d')`, где max_y_product – количество продукта произведённое при оптимальном плане, полученное из решения задачи 1

Матрица левых частей ограничений равенств:

```
A = matrix([[-0.005, 0, 0.003, -0.009, 0.003, 0],
            [0, -0.004, 0.003, 0, 0.003, -0.005]], tc='d')
```

Матрица правых частей ограничений равенств:

```
b = matrix([0, 0], tc='d')
```

Найдём решение задачи линейного программирования, вызвав функцию solution = solvers.lp(c, G.T, h, A.T, b, solver=’glpk’). После вызова данной функции, в solution должен находиться словарь, с параметрами, описанными в документации []. Нас интересует только некоторые из них:

- “status” – статус решения задачи линейного программирования
- “objective” – значение целевой функции
- “x” – оптимальный план
- “z” – массив с теневыми/приведёнными ценами (в зависимости от ограничений)

Результаты решения задачи при помощи библиотеки CVXOPT и внешнего решателя ‘glpk’ представлены на снимке экрана ниже (Рисунок 2).

```
Status: optimal
Objective: 285.2618483412323
x =
[ 7.45e+01]
[ 6.10e+01]
[ 7.50e+01]
[ 0.00e+00]
[ 4.91e+01]
[ 2.57e+01]

Shadow price=
[-0.00e+00]
[ 1.16e-01]
[ 6.55e-02]
[ 1.01e-01]

Reduced cost=
[-0.00e+00]
[-0.00e+00]
[-0.00e+00]
[ 2.83e-01]
[-0.00e+00]
[-0.00e+00]
```

Рисунок 2 – Результаты решения задачи распределения ресурсов

Для найденного решения задачи, проведём анализ чувствительности к изменению параметров правых частей неравенств часового фонда тех групп оборудования, которые загружены полностью. Для таких групп теневая цена не равна нулю. В нашем случае загружены полностью все три группы оборудования.

Приведем переработанный алгоритм поиска интервала допустимости решения [] для изменения правых частей:

1. Задаёмся постоянным шагом, с точностью до которого будем искать интервал допустимости.
2. Последовательно увеличивая правую часть на шаг и перерешивая задачу линейного программирования, находим момент, когда значение функции изменилось на число, отличное от теневой цены. Таким образом мы находим правую границу интервала допустимости.

3. Повторяем аналогичную процедуру в сторону уменьшения правой части и находим левую границу.
4. Полученные левая и правая границы – являются интервалом допустимости нашего решения.

```
Sensitivity analysis for y2 constraint
y2 availability interval is [300.0, 1278.0]

Sensitivity analysis for y3 constraint
y3 availability interval is [572.0, 4060.0]

Sensitivity analysis for y4 constraint
y4 availability interval is [267.0, 4300.0]
```

Рисунок 3 – Результат поиска интервалов допустимости для групп оборудования загруженных полностью

Исходный код анализа чувствительности приведен в приложении (Приложение А), результаты анализа чувствительности интерпретирует Рисунок 3 и Таблица 3

Таблица 3 – Интерпретация интервала допустимости решения в виде таблицы

Номер ограничения	Группа оборудования	Левая граница	Правая граница
1	-	-	-
2	1	300	1278
3	2	572	4060
4	3	267	4300

По полученным данным интервала допустимости, построим «заменитель» диаграммы Торнадо (Рисунок 4), показывающий, как изменяется целевая функция, на интервалах допустимости базисных переменных.

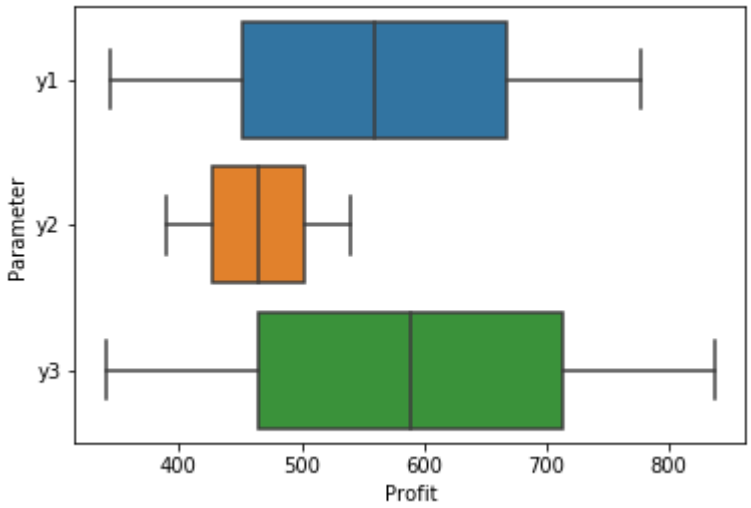

Рисунок 4 – «Диаграмма» Торнадо выполненная при помощи seaborn.boxplot

3. ТРАНСПОРТНАЯ ЗАДАЧА ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

3.1. Условие задачи

Аналогичные по функциональному назначению комплекты изделий производятся на трех других предприятиях (A_2-A_4) количествах 4400, 5900, 4200 комплектов. По 67% производимых на всех четырех предприятиях комплектов изделий перевозятся в пять городов (B_1-B_5), где данная продукция не производится, в количествах 1900,3200, 2900, 4100, 3500. Транспортные расходы на перевозку одного комплекта изделий представлены ниже (Таблица 4).

Таблица 4 – Транспортные расходы

	B1	B2	B3	B4	B5
A1	5.1 +	7.4	7.6	5.3 -	3.0
A2	5.6	7.4 -	4.0	7.9 +	6.6
A3	2.2	4.3 +	5.7 +	5.8 -	6.6
A4	5.1 +	5.3	3.3	6.7	6.8

Однако, следует иметь в виду, что цены доставки являются приближенными, причем тенденции изменения некоторых удельных стоимостей перевозок обозначены в Таблица 4 («-» - уменьшение, «+» - увеличение). При решении транспортной задачи предусмотреть различные варианты планирования перевозок в зависимости от вида несбалансированности задачи:

1. Если имеется избыток запасов, то предполагается организовать дополнительное потребление, причем:

- Если избыток не превышает 15%, то потребление сосредоточивается в одном дополнительном пункте (соответствует решению задачи с фиктивным пунктом назначения)

- Если избыток превышает 15%, то увеличивается потребление в каждом пункте назначения (соответствует распределению грузов между пунктами назначения пропорционально заявкам)

2. При недостатке запасов в зависимости от его величины
    - При недостатке запасов свыше 15% формируется дополнительная заявка от неудовлетворенных потребителей (соответствует решению задачи с фиктивным пунктом отправления);
    - При недостатке запасов менее 15 % грузы распределяются между пунктами назначения пропорционально заявкам.

Требуется:

1. Найти план перевозок, оптимальный по критерию стоимости;
2. Сформулировать рекомендации по результатам решения транспортной задачи в зависимости от вида несбалансированности задачи;
3. Исследовать решение на чувствительность к изменению целевой функции в зависимости от возможного изменения цен.

3.2. Формализация задачи

Введём обозначения:

- x_{ij} – количество перевезённого продукта i в пункт j
- $prev$ – количество произведённой продукции заводом A_1, вычисленное в предыдущей задаче

Основной целью задачи является минимизация расходов на перевозки, следовательно, составим целевую функцию, непосредственно интерпретируя условие задачи (Таблица 4):

$$f(\vec{x}) = 5.1x_{11} + 7.4x_{12} + 7.6x_{13} + 5.3x_{14} + 3.0x_{15} + 5.6x_{21} + 7.4x_{22} + 4.0x_{23} + 7.9x_{24} + 6.6x_{25} + 2.2x_{31} + 4.3x_{32} + 5.7x_{33} + 5.8x_{34} + 6.6x_{35} + 5.1x_{41} + 5.3x_{42} + 6.7x_{44} + 6.8x_{45} \rightarrow min$$

Так как количество продукции, производимое всеми заводами ограничено и оговорено в задании, составим ограничения на количество продукции:

$$\sum_{j=1}^{5} x_{1j} \leq 0.67 \cdot prev$$

$$\sum_{j=1}^{5} x_{2j} \leq 0.67 \cdot 4400$$

$$\sum_{j=1}^{5} x_{3j} \leq 0.67 \cdot 5900$$

$$\sum_{j=1}^{5} x_{4j} \leq 0.67 \cdot 4200$$

Так как объемы потребления продукции ограничено и оговорено в задании, составим ограничения на потребление продукции:

$$\sum_{i=1}^{4} x_{i1} = 1900$$

$$\sum_{i=1}^{4} x_{i2} = 3200$$

$$\sum_{i=1}^{4} x_{i3} = 2900$$

$$\sum_{i=1}^{4} x_{i4} = 4100$$

$$\sum_{i=1}^{4} x_{i5} = 3500$$

Так как по условию задачи потребление не может быть отрицательным, введём ограничения неотрицательности переменных:

$$x_{ij} \geq 0, i = 1..4, j = 1..5$$

Все ограничения, а также целевая функция - линейны, все переменные – неотрицательны, следовательно перед нами задача линейного программирования.

3.2. Решение задачи

Для начала проверим, количество ресурсов доступных для перевозки, для этого сложим все правые части ограничений на количество продукций и вычтем сумму всех правых частей ограничений на потребление продукций:

```
manufactured = [0.67 * i for i in [max_product, 4400, 5900, 4200]]
required = [1900, 3200, 2900, 4100, 3500]
delta = sum(required) - sum(manufactured)
print(f"Manufactured = {sum(manufactured)}, required = {sum(required)}, delta =
{sum(required) - sum(manufactured)}")
```

Результат выполнения кода представлен ниже (Рисунок 5). Мы видим, что недостаток производства ресурсов ~5693 единиц, следовательно решаем задачу с фиктивным пунктом отправления грузов с ограничением на количество продукции с правой частью 5693. Новое ограничение на количество продукции выглядит следующим образом:

$$\sum_{j=1}^{5} x_{5j} \leq 5693$$

Так как пункт отправки – фиктивный, на целевой функции он никак не отражается. В ограничениях на потребление, к каждому пункту добавляем также часть ресурсов, доставляемых новым фиктивным пунктом отправления. Пример

нового ограничения для потребления для пункта потребления B2 представлен ниже.

$$\sum_{i=1}^{5} x_{i2} = 3200$$

Также, фиктивный пункт производства, не может доставлять отрицательное количество ресурсов, следовательно, новое ограничение неотрицательности будет выглядеть следующим образом:

$$x_{ij} \geq 0, i = 1..5, j = 1..5$$

Решение задачи линейного программирования при помощи CVXOPT описано в прошлой главе, Рисунок 5 представляет результат выполнения программы, Таблица 5 показывает оптимальный план для перевозок при текущем положении цен.

Таблица 5 – Оптимальный по стоимости план перевозок

	B1	B2	B3	B4	B5
A1	0	0	0	0	191
A2	1900	0	0	1050	0
A3	0	0	2900	1050	0
A4	0	0	0	2000	815
A5	0	3200	0	0	2490

В таблице выше (Таблица 5) видно, что наибольший дефицит продукта находится в пунктах B2 и B5, рекомендуется увеличивать фонд рабочих часов для оборудования из прошлой задачи, так как, имеется большое количество

нереализованного продукта и недостаток в 5693 единицы продукта в пунктах потребления

```
Manufactured = 9906.125438388626, required = 15600, delta = 5693.874561611374
Solution status = optimal
Minimal cost = 41554.8
Plan =
 [ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 1.91e+02]
[ 1.90e+03]
[ 0.00e+00]
[ 0.00e+00]
[ 1.05e+03]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 2.90e+03]
[ 1.05e+03]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 2.00e+03]
[ 8.15e+02]
[ 0.00e+00]
[ 3.20e+03]
[ 0.00e+00]
[ 0.00e+00]
[ 2.49e+03]
```

Рисунок 5 – Результат поиска оптимального решения для транспортной задачи линейного программирования

Исследуем решение на чувствительность к изменению целевой функции [] в зависимости от возможного изменения функций, отображённого в условии задачи (Таблица 4). Код, реализующий анализ чувствительности целевой функции к изменению коэффициентов представлен в приложении (Приложение В), результат анализа чувствительности представлен ниже (Рисунок 6 и Таблица 6).

```
x11 variable is out of plan and increases, skipping.
x15 variable decreases, will be in plan with delta=-0.1
x22 variable decreases, will be in plan with delta=-3.399999999999996
x24 variable increases, will be out of plan with delta=2.8999999999999897
x32 variable is out of plan and increases, skipping.
x33 variable increases, will be out of plan with delta=4.599999999999992
x34 variable decreases, will be in plan with delta=-0.09999999999999964
x41 variable is out of plan and increases, skipping.
```

Рисунок 6 – Анализ чувствительности целевой функции

Таблица 6 – Интерпретация результатов анализа чувствительности ЦФ

Переменная	Направление изменения	Присутствие в базисе	Изменение коэффициента для смены присутствия в базисе
x_{11}	+	-	-
x_{15}	-	-	-0.1
x_{22}	-	-	-3.4
x_{24}	+	+	2.9
x_{32}	+	-	-
x_{33}	+	+	4.6
x_{34}	-	-	-0.1
x_{41}	+	-	-

ЗАКЛЮЧЕНИЕ

В результате данной работы были получены практические навыки решения линейных и динамических оптимизационных задач. Были исследованы методы решения и оптимизации времени исполнения решений оптимизационных задач. Ко всем задачам были составлены формальные математические модели и приведены решения. По результатам решения задач, были составлены естественные интерпретации и приведены советы.

СПИСОК ЛИТЕРАТУРЫ

2. Extending Python with C or C++ // Python Documentation. URL: https://docs.python.org/3.7/extending/extending.html (дата обращения: 16.04.2019).
3. Function Call Overhead // Python Performance Tips. URL: https://nyu-cds.github.io/python-performance-tips/04-functions/ (дата обращения: 16.04.2019).
4. Документация CVXOPT [Электронный ресурс] // CVXOPT: [сайт]. URL: https://cvxopt.org/documentation/index.html (дата обращения: 16.04.2019).
5. Пономарев А.В. Динамическое программирование с помощью GNU Octave за 7 простых шагов [Электронный ресурс] // Теория принятия решений - тематический сайт: [сайт]. URL: http://cais.iias.spb.su/ponomarev/DP_Octave.pdf (дата обращения: 15.04.2019).
6. Пономарёв А.В. Анализ чувствительности в задачах ЛП с помощью Python // Теория принятия решений - тематический сайт. URL: https://avponomarev.bitbucket.io/slides/%D0%9A%D1%83%D1%80%D1%81%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B2%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D1%87%D1%83%D0%B2%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%9B%D0%9F.pdf (дата обращения: 16.04.2019).
7. Пономарёв А.В. Анализ чувствительности: общие идеи // Теория принятия решений - тематический сайт. URL: https://avponomarev.bitbucket.io/slides/%D0%9A%D1%83%D1%80%D1%81%D0%BE%D0%B2%D0%B0

%D1%8F_%D0%B2%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D1%87%D1%83%D0%B2%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BE%D0%B1%D1%89%D0%B5%D (дата обращения: 16.04.2019).

8. Пономарёв А.В. Динамическое программирование (задача о рюкзаке) // Теория принятия решений - тематический сайт. URL: https://avponomarev.bitbucket.io/slides/%D0%9A%D1%83%D1%80%D1%81%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B2%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%94%D0%9F_%D1%80%D1%8E%D0%BA%D0%B7%D0%B0%D0%BA.pdf (дата обращения: 16.04.2019).
9. Пономарёв А.В. Динамическое программирование (Реализация) // Теория принятия решений - тематический сайт. URL: https://avponomarev.bitbucket.io/slides/%D0%9A%D1%83%D1%80%D1%81%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B2%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%94%D0%9F_%D1%80%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F.pdf (дата обращения: 16.04.2019).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД АНАЛИЗА ЧУВСТВИТЕЛЬНОСТИ ДЛЯ ПРАВЫХ ЧАСТЕЙ ОГРАНИЧЕНИЙ

```
from typing import Tuple


def availability_interval(c, G, h, A, b, constraint_index: int) -> Tuple[int,
int]:
    """
    Returns solution availability interval for constraint with defined index

    :param c: default cvxopt.solvers.lp argument
    :param G: default cvxopt.solvers.lp argument
    :param h: default cvxopt.solvers.lp argument
    :param A: default cvxopt.solvers.lp argument
    :param b: default cvxopt.solvers.lp argument
    :param constraint_index: index of constraint
    :return: availability interval for constraint
    """
    dh = matrix([int(i == constraint_index) for i in range(len(h))], tc='d')
    default_solution = solvers.lp(c, G.T, h, A.T, b, solver='glpk')

    price = default_solution['z'][constraint_index]
    prev_z = -default_solution['primal objective']

    step = 100
    a = 1

    while True:
        solution = solvers.lp(c, G.T, h + dh * a, A.T, b, solver='glpk')

        if solution['status'] != 'optimal':
            return 0, 0

        new_z = -solution['primal objective']
        delta_z = new_z - prev_z
        prev_z = new_z

        if abs(delta_z - price) > 1e-6:
            right_border = h[constraint_index] + a
            break

        a += 1

    a = 1
    prev_z = -default_solution['primal objective']

    while True:
        solution = solvers.lp(c, G.T, h - dh * a, A.T, b, solver='glpk')
```

```
        if solution['status'] != 'optimal':
            return 0, 0

        new_z = -solution['primal objective']
        delta_z = prev_z - new_z
        prev_z = new_z

        if abs(delta_z - price) > 1e-6:
            left_border = h[constraint_index] - a
            break

        a += 1

    return left_border, right_border

borders = {}

# Calculate availability interval for all constraints with non-zero shadow price
for constraint_index, price in enumerate(shadow_price):
    if price == 0:
        continue
    print(f'Sensitivity analysis for y{constraint_index + 1} constraint')
    left_border,  right_border  =  availability_interval(c,  G,  h,  A,  b,
constraint_index)
    borders[constraint_index] = (left_border, right_border)
    print(f'y{constraint_index  +  1}  availability  interval  is  [{left_border},
{right_border}]')
    print()
```

ПРИЛОЖЕНИЕ В

АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ ДЛЯ КОЭФФИЦИЕНТОВ ЦЕЛЕВОЙ ФУНКЦИИ

```
# Sensitivity analysis for target-function coefficients
from typing import Optional


def availability_interval_for_coeffs(
        c,
        G,
        h,
        A,
        b,
        coefficient_index: int,
        increase: bool,
        delta: float
) -> Optional[float]:
    """

    :param c: cvxopt.solvers.lp param
    :param G: cvxopt.solvers.lp param
    :param h: cvxopt.solvers.lp param
    :param A: cvxopt.solvers.lp param
    :param b: cvxopt.solvers.lp param
    :param coefficient_index: target function coefficient index
    :param increase:
    :param delta:
    :return:
    """
    solution = solvers.lp(c, G.T, h, A.T, b, solver='glpk')
    new_c = matrix([i for i in c], tc='d')

    if increase:
        # Check for absence of current coefficient index in optimal plan
        if solution["x"][coefficient_index] == 0:
            return

        while solution["x"][coefficient_index] > 0:
            new_c[coefficient_index] += delta
            solution = solvers.lp(new_c, G.T, h, A.T, b, solver='glpk')
    else:
        # Check for presence of current coefficient index in optimal plan
        if solution["x"][coefficient_index] > 0:
            plan_delta = None

            while plan_delta != 0:
                new_c[coefficient_index] -= delta
                new_solution = solvers.lp(new_c, G.T, h, A.T, b, solver='glpk')
```

```
                plan_delta     =     new_solution["x"][coefficient_index]     -
solution["x"][coefficient_index]
                solution = new_solution
        else:
            while solution["x"][coefficient_index] == 0:
                new_c[coefficient_index] -= delta
                solution = solvers.lp(new_c, G.T, h, A.T, b, solver='glpk')

    return new_c[coefficient_index] - c[coefficient_index]


# Variables change tendencies are defined in following manner
#
# True if variable may increase
# None if variable is stable
# False if variable may decrease

variables_tendencies = [
    True, None, None, None, False,
    None, False, None, True, None,
    None, True, True, False, None,
    True, None, None, None, None,
]

# Output all sensitivity analysis results corresponding to variables tendencies

for index, tendency in enumerate(variables_tendencies):
    variable_name = f'x{index // 5 + 1}{index % 5 + 1}'
    prefix = f'{variable_name} variable'

    if tendency is None:
        pass

    elif tendency:
        delta = availability_interval_for_coeffs(c, G, h, A, b, index, tendency,
0.1)

        if delta is None:
            print(f'{prefix} is out of plan and increases, skipping.')
        else:
            print(f'{prefix} increases, will be out of plan with delta={delta}')

    else:
        delta = availability_interval_for_coeffs(c, G, h, A, b, index, tendency,
0.1)

        if delta is None:
            print(f'{prefix} decreases and is present in plan, decreasing will
be useless with delta={delta}')
        else:
            print(f'{prefix} decreases, will be in plan with delta={delta}')
```