Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе No6

По теме “Задачи квадратичного программирования”

Выполнила: студентка гр. 053503 Зырянова М.М.

Проверил: ассистент кафедры информатики Туровец Н. О.

Минск 2023

**СОДЕРЖАНИЕ**

[1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 3](#_Toc133432474)

[2 АЛГОРИТМ ДВОЙСТВЕННОГО СИМПЛЕКС МЕТОДА 4](#_Toc133432475)

[3 ПРИМЕР РЕАЛИЗАЦИИ ДВОЙСТВЕННОГО СИМПЛЕКС МЕТОДА 5](#_Toc133432476)

[4 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ 9](#_Toc133432477)

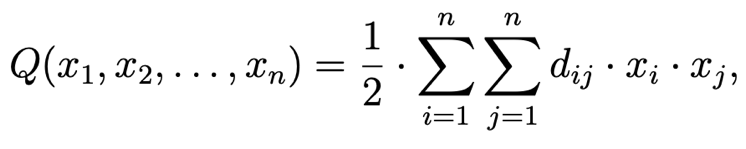
[ПРИЛОЖЕНИЕ А 10](#_Toc133432478)

# **1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Определение 1. Квадратная матрица D = (dij) порядка n называется симметричной, если D = D′ (dij = dji для любых 1 ≤ i, j ≤ n).

Определение 2. Квадратная матрица D = (dij) порядка n называет- ся положительно полуопределённый (или неотрицательно определён- ной), если для всякого вектора x ∈ Rn имеет место неравенство x′·D·x ≥ 0.

Определение 3. Пусть x1, x2, . . . , xn ∈ R — переменные. Выражение



для которого выполняется условие dij = dji для любых 1 ≤ i, j ≤ n, на- зывается квадратичной формой. Квадратная матрица D, составленная из коэффициентов dij, является симметричной.

В матричном виде квадратичная форма записывается так

Q(x) = 21 · x′ · D · x,

где x′ = (x1, x2, . . . , xn) — вектор переменных.

Задача квадратичного программирования — это оптимизационная за-

дача вида

f(x)=c′ ·x+21·x′ ·D·x→min,

A · x = b,

x ≥ 0,

в которой матрица D является симметричной положительно полуопре- делённой.

Определение 4. Допустимый план x задачи квадратичного програм- мирования называется правильным опорным планом, если существу- етподмножествоJb множестваиндексовпеременных{1,2,...,n}такое, что

1. |Jb| = rank(A);

2. матрица Ab, составленная из столбцов матрицы A, чьи индексы

принадлежат множеству Jb, является обратимой;

3. Найдётся подможество Jb⋆ множества индексов переменных {1, 2, . . . , n} такое, что

(a) Jb ⊆Jb⋆;

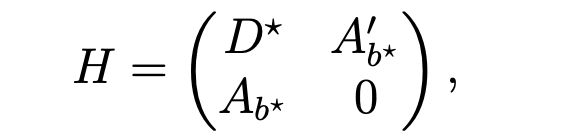
(b) для каждого индекса j ∈ Jb⋆ выполняется ∆j (x) = 0, где

c′(x) = c′ +x′ ·D (1)

u′(x) = −c′ (x) · A−1 (2)

∆′(x) = u′(x) · A + c′(x) (3)

(c) следующая блочная матрица обратима

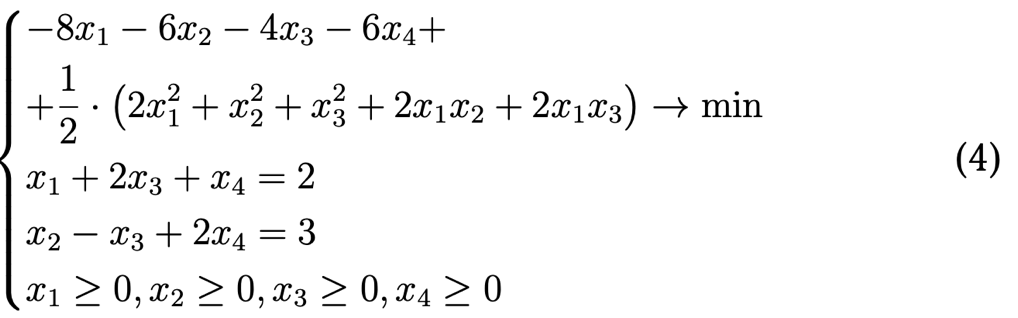


где D⋆ — это подматрица матрицы D, составленная из элемен- тов, стоящих на пересечении строк и столбцов с индексами из множества Jb⋆ ; Ab⋆ — матрица, состоящая из столбцов матри- цы A с индексами из множества Jb⋆.

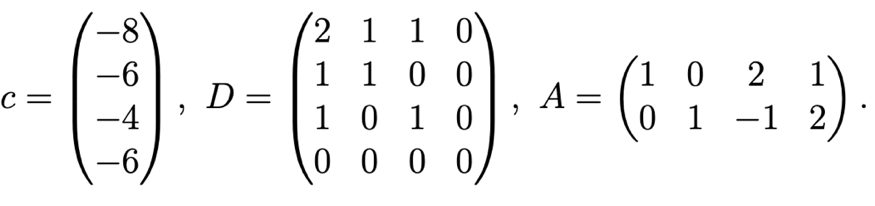
Множество Jb называется опорой ограничений, а множество Jb⋆ — рас- ширенной опорой ограничений.

# **2** **ПРИМЕР РЕШЕНИЯ ЗАДАЧИ КВАДРАТИЧНОГО ПРОГРАММИРОВАНИЯ**

Рассмотрим следующую задачу квадратичного программирования:



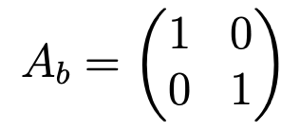
Покажем, что допустимый план x′ = (2, 3, 0, 0) этой задачи является правильным опорным планом с опорой ограничений Jb = {1,2} и расширенной опорой ограничений Jb⋆ = {1,2}. Составим матрицы A,D и вектор c:



Проверим условия из определения правильного опорного плана:

1. Поскольку |Jb| = 2 и rank(A) = 2, то |Jb| = rank(A).

2. Матрица

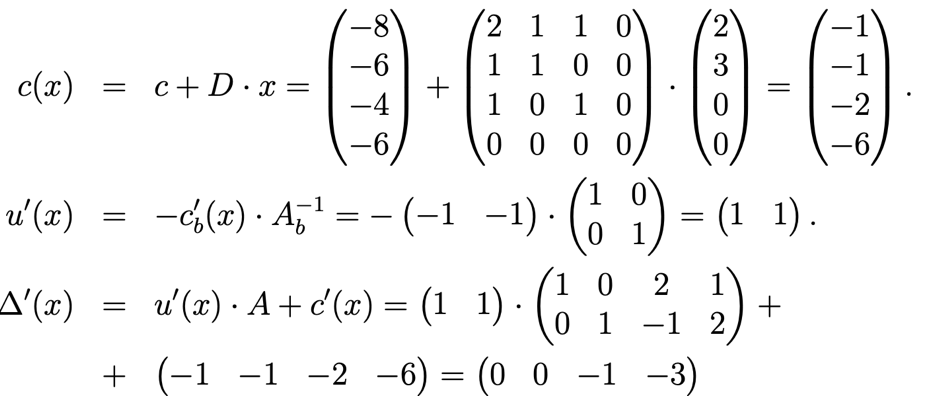


невырожденная.

3. Проверим условия для Jb⋆

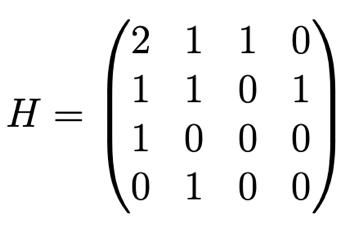
(a) {1,2} = Jb ⊆ Jb⋆ = {1,2};

(b) Вычислим векторы c(x), u(x) и ∆(x):



Компоненты вектора ∆′(x) с индексами из множества Jb⋆ не меньше, чем 0.

(c) Матрица



имеет определитель, отличный от 0, т.е. она обратима.

В рамках лабораторной работы необходимо реализовать метод решения задач квадратичного программирования, работу которого проиллюстри- руем на примере.

**Пример**. Рассмотрим задачу квадратичного программирования (4), взяв в качестве начального правильного опорного плана план x′ = (2, 3, 0, 0) с опорой ограничений Jb = {1,2} и расширенной опорой ограничений Jb⋆ = {1, 2}.

**Итерация 1**. ШАГ 1. Находим векторы c(x),u(x) и ∆(x) по формулам (1) – (3). Эти векторы мы уже нашли, когда проверяли выполнимость условий из определения правильного опорного плана для плана x. Приведём результаты:

c′(x) = (−1 −1 −2 −6),

u′(x) = (1 1),

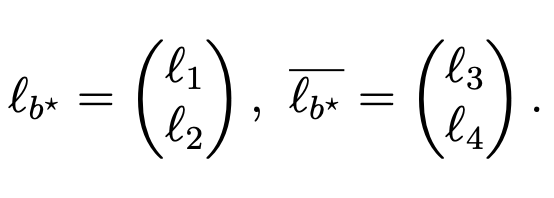
∆′(x) = (0 0 −1 −3).

ШАГ 2. Проверяем условие оптимальности текущего правильного опорного плана. Если все компоненты вектора ∆(x) неотрицательные, то метод завершает свою работу и текущий правильный опорный план является оптимальным. В противном случае переходим на следующий шаг.

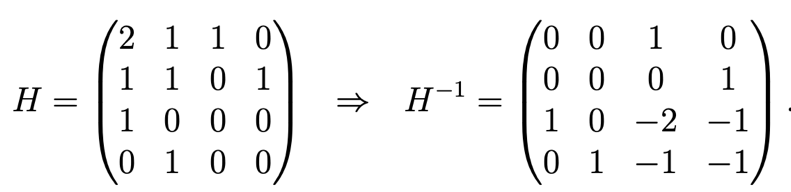
ШАГ 3. Выбираем отрицательную компоненту плана ∆(x) и индекс выбранной компоненты запоминаем в переменной j0. Пусть j0 = 3.

ШАГ 4. По множеству Jb⋆ и j0 найдём вектор l′ = (l1, l2, l3, l4). Ком-

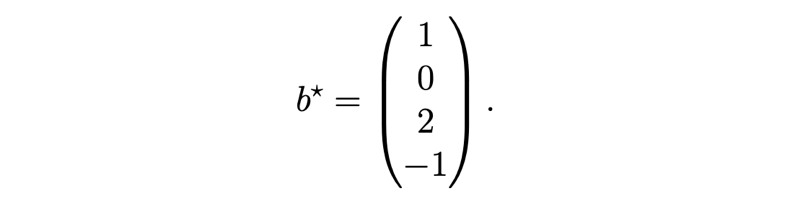
поненты этого вектора делятся на два класса



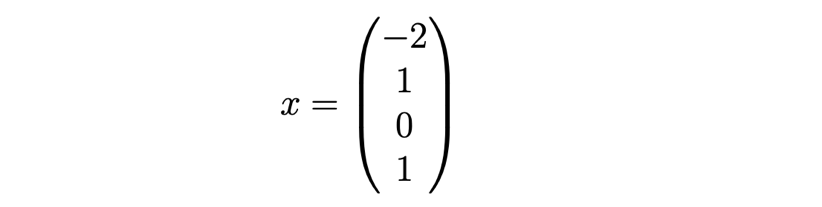
К первому классу относятся все компоненты с индексами из расширен- ной опоры ограничений Jb⋆ , ко второму — все компоненты с индексами не из расширенной опоры ограничений. Сперва находим вектор lb⋆ по следующему правилу: lj0 = 1, значения всех остальных компонент lb⋆ полагаем равными 0. В нашем случае l3 = 1 и l4 = 0. Для того, чтобы найти компоненты вектора lb⋆ мы составляем матрицу H (см. определе- ние правильного опорного плана) и обращаем её



Строим вектор b⋆. Он состоит из двух частей. Сперва идут элементы столбца матрицы D с индексом j0, стоящие в строках с индексами из множества Jb⋆. Далее идут элементы j0-го столбца матрицы A. В нашем случае



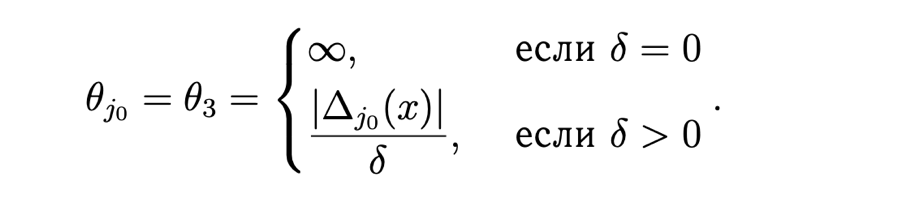
Находим вектор x по следующей формуле x = −H−1 · b∗. Получаем



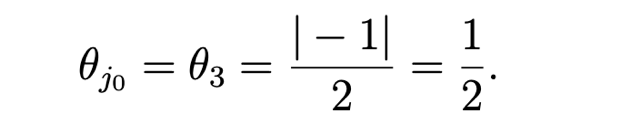
Сколько компонент в искомом векторе lb⋆ ? Две компоненты. Первые две компоненты вектора x и есть компоненты вектора lb⋆ : l1 =−2, l2 =1. В итоге получаем l′ = (−2 1 1 0).

ШАГ 5. Для каждого индекса j ∈ Jb⋆ найдём величину θj, а также

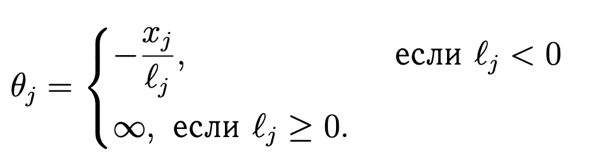
вычислим величину θj0 . Сперва поищем θj0 . Для этого вычислим значение δ = l′ · D · l = 2. Затем найдём θj0 по следующему правилу



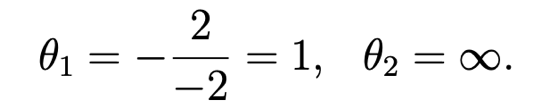
Поскольку δ = 2 > 0, то в нашем случае реализовался второй случай, в соответствии с которым



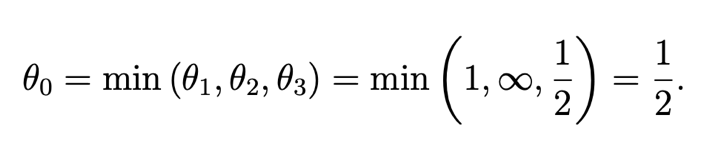
Для каждого индекса j ∈ Jb⋆ вычислим



В нашем случае получаем, что

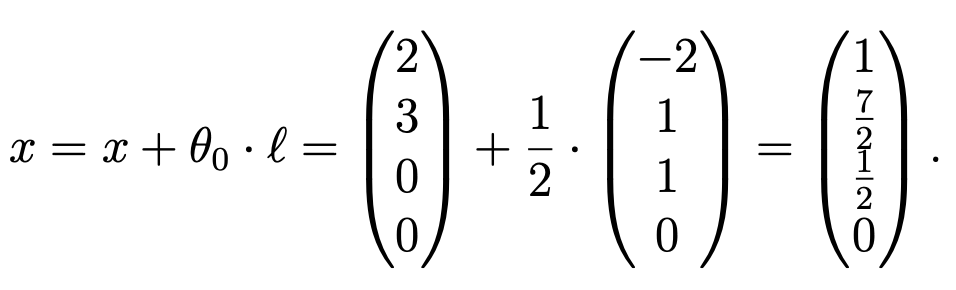


Находим минимум среди всех вычисленных θk:



Запоминаем индекс, на котором достигается минимум j⋆ = 3. Если θ0 = ∞, то метод завершает свою работу с ответом: «целевая функция задачи не ограничена снизу на множестве допустимых планом». В противном случае идём на следующий шаг.

ШАГ 6. Обновим допустимый план



Обновим теперь опору ограничений Jb и расширенную опору ограни- чений Jb⋆ по следующему правилу:

1. Если j⋆ = j0, то Jb не меняем, а в Jb⋆ добавляем j⋆.  
2. Еслиj⋆ ∈Jb⋆ \Jb,тоJb неменяем,аизJb⋆ удаляемj⋆.

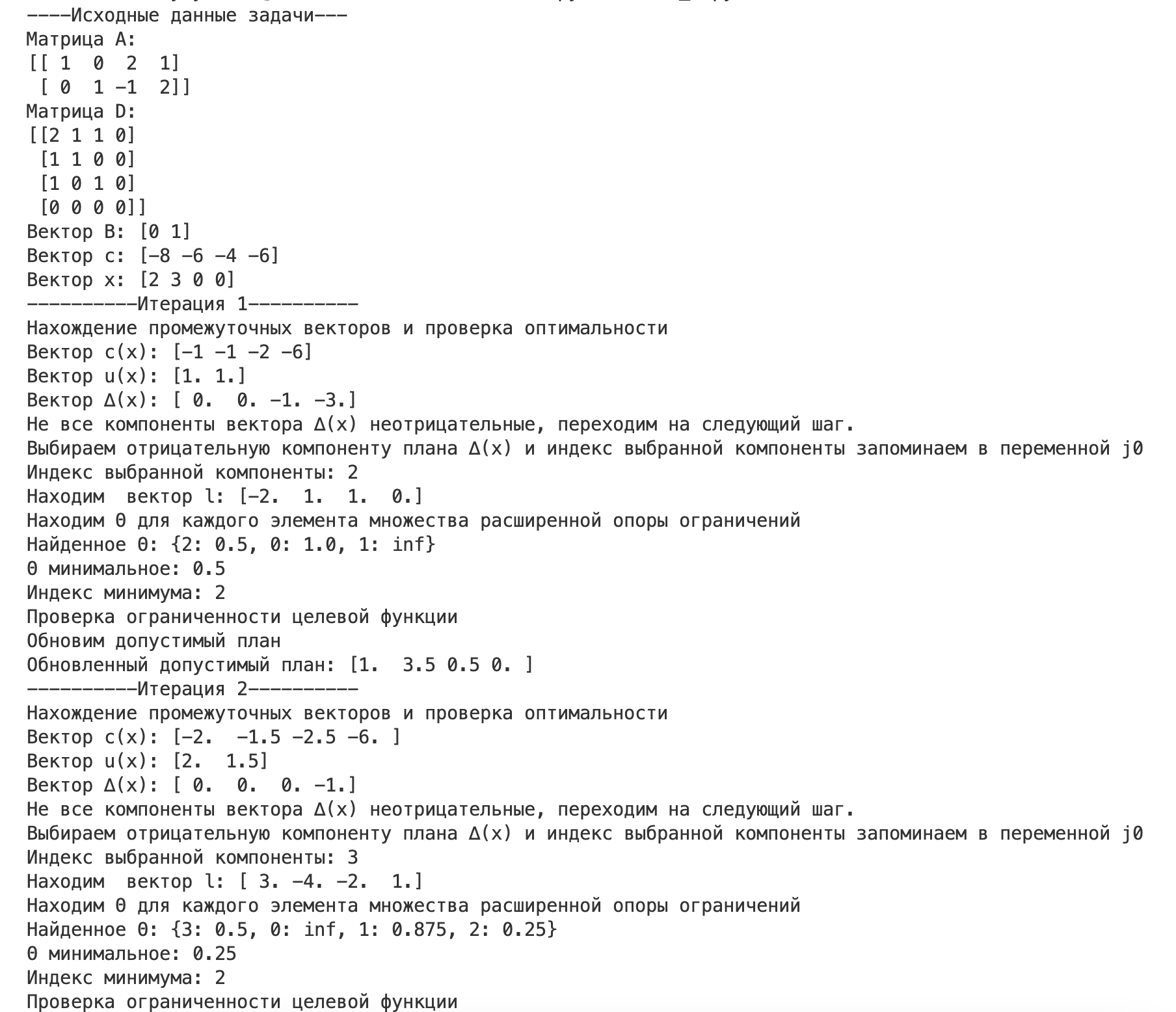
3. Если j⋆ ∈ Jb (индекс j⋆ идёт s-м по счёту в Jb) и существует индекс j+ ∈ Jb⋆ \ Jb такой, что s-я компонента вектора (A−1 · Aj+ ) не равна 0, то в Jb заменяем индекс j⋆ на индекс j+, а из Jb⋆ удаляем индекс j⋆.

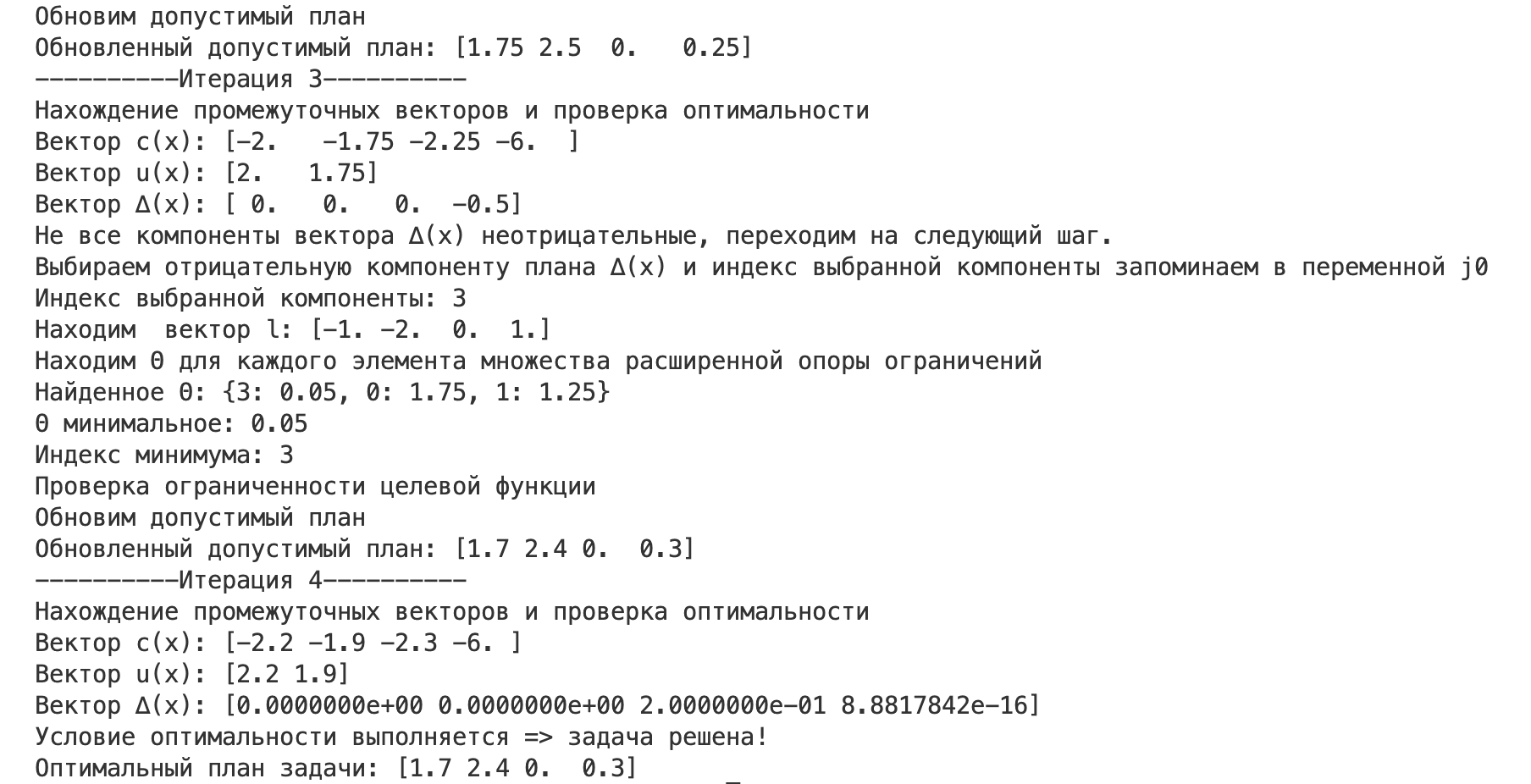
4. Если j⋆ ∈Jb (индекс j⋆ идёт s-м по счёту в Jb) и (Jb =Jb⋆ или для любого индекса j+ ∈ Jb⋆ \ Jb выполняется условие: s-ая компонента вектора (A−1 · Aj+ ) равна 0), то и в Jb, и в Jb⋆ заменяем индекс j⋆ b на индекс j0.

В нашем случае j⋆ = j0, т.е. реализовался первый случай. В соот- ветствии с правилом Jb = {1,2} не меняется, а в Jb⋆ добавляем j⋆ и получаем Jb⋆ = {1, 2, 3}.

С новым правильным опорным планом x, новыми множествами ин- дексов Jb, Jb⋆ идём на следующую итерацию, на котором проделываем все те же действия, что и на первой итерации.

# **4 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ**





# **ПРИЛОЖЕНИЕ А**

(Листинг программы)

import numpy as np

def get\_basis\_vector(c, B):

i = 0

c\_b = [0 for \_ in B]

for index in B:

c\_b[i] = c[index]

i += 1

return c\_b

def check(delta\_x):

for i in range(len(delta\_x)):

if delta\_x[i] < 0:

return i

return -1

if \_\_name\_\_ == "\_\_main\_\_":

c = np.array([-8, -6, -4, -6])

A = np.array([[1, 0, 2, 1],

[0, 1, -1, 2]])

x = np.array([2, 3, 0, 0])

B = np.array([0, 1])

Bs = np.array([0, 1])

D = np.array([[2, 1, 1, 0],

[1, 1, 0, 0],

[1, 0, 1, 0],

[0, 0, 0, 0]])

counter = 0

print('----Исходные данные задачи---')

print('Матрица A: ', A, sep='\n')

print('Матрица D: ', D, sep='\n')

print('Вектор B: ', B, sep='')

print('Вектор с: ', c, sep='')

print('Вектор x: ', x, sep='')

while True:

counter += 1

print(f'----------Итерация {counter}----------')

A\_b = A[:, B]

A\_b\_inv = np.linalg.inv(A\_b)

print('Нахождение промежуточных векторов и проверка оптимальности')

c\_x = c + np.dot(x, D)

c\_b = get\_basis\_vector(c\_x, B)

c\_b = [i\*(-1) for i in c\_b]

u\_x = np.dot(c\_b, A\_b\_inv)

delta\_x = np.dot(u\_x, A) + c\_x

print('Вектор с(x): ', c\_x, sep='')

print('Вектор u(x): ', u\_x, sep='')

print('Вектор ∆(x): ', delta\_x, sep='')

j0 = check(delta\_x)

if j0 == -1:

print('Условие оптимальности выполняется => задача решена!')

print('Оптимальный план задачи: ', x, sep='')

break

print('Не все компоненты вектора ∆(x) неотрицательные, переходим на следующий шаг.')

print('Выбираем отрицательную компоненту плана ∆(x) и индекс выбранной компоненты запоминаем в переменной j0')

print('Индекс выбранной компоненты: ', j0, sep='')

l = np.zeros(len(x))

l[j0] = 1

A\_b\_ext =A[:, Bs]

H = np.bmat([

[D[Bs, :][:, Bs], A\_b\_ext.T],

[A\_b\_ext, np.zeros((len(A), len(A)))]

])

H\_inv = np.array(np.linalg.inv(H))

b\_starred = np.concatenate((D[Bs, j0], A[:, j0]))

x\_temp = np.dot(-H\_inv, b\_starred)

l[:len(Bs)] = x\_temp[:len(Bs)]

print('Находим вектор l: ', l, sep='')

print('Находим Θ для каждого элемента множества расширенной опоры ограничений')

sigma = np.dot(np.dot(l, D), l)

theta = {}

theta[j0] = np.inf if sigma == 0 else np.abs(delta\_x[j0]) / sigma

for j in Bs:

if l[j] < 0:

theta[j] = -x[j] / l[j]

else:

theta[j] = np.inf

j\_s = min(theta, key=theta.get)

theta\_0 = theta[j\_s]

print('Найденное Θ: ', theta, sep='')

print('Θ минимальное: ', theta\_0, sep='')

print('Индекс минимума: ', j\_s, sep='')

print('Проверка ограниченности целевой функции')

if theta\_0 == np.inf:

print('Функция не ограничена множеством допустимых планов')

print('Обновим допустимый план')

x = x + theta\_0 \* l

print('Обновленный допустимый план: ', x, sep='')

if j\_s == j0:

Bs = np.append(Bs, j\_s)

elif j\_s in Bs and j\_s not in B:

Bs = np.delete(Bs, j\_s)

elif j\_s in B:

third\_condition = False

s = B.index(j\_s)

print('Обновляем опоры ограничений')

for j\_plus in set(Bs).difference(B):

if (np.dot(A\_b\_inv, A[:, j\_plus]))[s] != 0:

third\_condition = True

B[s] = j\_plus

Bs = np.delete(Bs, j\_s)

if not third\_condition:

B[s] = j0

Bs[Bs.index(j\_s)] = j0

print('Обновленные опоры ограничений: ', Bs, sep='')