ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ РФ

ГОУ ВПО «Поволжская государственная
академия телекоммуникаций
и информатики»

В.Н. Тарасов, Н.Ф. Бахарева

МАТЕМАТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ ТЕОРИЯ АЛГОРИТМЫ ПРОГРАММЫ

ИЗДАНИЕ ВТОРОЕ
ПЕРЕРАБОТАННОЕ

Допущено Научно-методическим советом по математике Министерства образования и науки Российской Федерации в качестве учебного пособия для студентов, обучающихся по направлению «Прикладная математика и информатика» (010500).

Самара 2007

ББК 22.18я7
Т 19
УДК [517.8+004.421](075.8)

Рецензенты: заведующий кафедрой информационных систем и технологий СГАУ, Заслуженный работник высшей школы РФ, Академик международной академии информатизации, д.т.н., профессор **С.А.Прохоров**; заведующий кафедрой теории оптимального управления МЭСИ, д.э.н., профессор **Б.А.Лагоша**; заведующий кафедрой прикладной математики ГУУ, д.э.н., профессор **В.А.Колемаев**.

Тарасов В.Н., Бахарева Н.Ф.

Математическое программирование. Теория, алгоритмы, программы. – Самара: РИЦ «Гольфстрим», 2007. –222 с.
Т 19
ISBN 5-7410-0559-4

Учебное пособие предназначено для студентов специальностей по направлениям «Прикладная информатика (по областям)», «Информатика и вычислительная техника» и «Информационные технологии», изучающих дисциплину «Методы оптимизации».

ББК 22.18я7

ISBN 5-7410-0559-4

Содержание

	Введение	6
1	Линейное программирование	8
1.1	Примеры задач линейного программирования	8
1.1.1	Задача планирования выпуска продукции (планирование производства)	8
1.1.2	Планирование капитальных вложений	10
1.2	Основные определения	10
1.3	Геометрическая интерпретация двумерной задачи линейного программирования и ее решение	13
1.3.1	Свойства задачи линейного программирования	17
1.3.2	Обоснование симплекс метода	19
1.3.3	Нахождение начального базиса	25
1.3.4	Решение в форме симплекс-таблиц	28
1.4	Двойственная задача линейного программирования	32
1.4.1	Пример прямой и двойственной задачи линейного программирования	32
1.4.2	Общая формулировка прямой и двойственной задачи	34
1.4.3	Свойства двойственной задачи	36
1.4.4	Анализ чувствительности	40
1.4.5	Экономическая интерпретация двойственных задач	46
2	Специальные задачи линейного программирования	49
2.1	Транспортная задача	49
2.2	Поиск начального опорного плана	53
2.2.1	Метод северо-западного угла	53
2.2.2	Метод минимального элемента	55
2.2.3	Решение транспортной задачи методом потенциалов	56
2.3	Анализ чувствительности	61
3	Дискретное программирование	63
3.1	Задачи целочисленного линейного программирования	63
3.1.1	Задача о размещениях	63

3.1.2 Задача о назначениях 64
3.1.3 Задача о коммивояжере 65
3.2 Методы решения задач целочисленного программирования 66
3.2.1 Метод отсечения Гомори 66
3.2.2 Метод ветвей и границ 69
3.2.3 Метод ветвей и границ решения задачи о коммивояжере 70
3.2.4 Аппроксимация решения задачи о коммивояжере 75
4 Методы безусловной оптимизации 77
4.1 Численные методы безусловной минимизации функции одной переменной. Методы прямого линейного поиска 77
4.1.1 Метод равномерного поиска 79
4.1.2 Метод золотого сечения 79
4.2 Численные методы безусловной минимизации функции многих переменных 81
4.2.1 Методы многомерного прямого поиска 81
4.2.2 Метод циклического покоординатного спуска 82
4.2.3 Метод Хука-Дживса 85
4.2.4 Метод наискорейшего спуска 85
5 Задачи нелинейного программирования 88
5.1 Постановка задачи и основные определения 88
5.2 Геометрическая интерпретация решения задач нелинейного программирования 90
5.3 Задачи выпуклого программирования 93
5.3.1 Основные определения и теоремы 93
5.3.2 Метод неопределенных множителей Лагранжа для решения задач квадратичного программирования 98
5.4 Градиентные методы решения задач нелинейного программирования 101
5.4.1 Метод приведенного градиента Вулфа 102
5.4.2 Метод штрафных функций 109
6 Лабораторные работы 114
6.1 Лабораторная работа №1. Решение задачи линейного программирования 114

6.2 Лабораторная работа №2. Решение двойственной задачи линейного программирования 126
6.3 Лабораторная работа №3. Решение транспортной задачи 130
6.4 Лабораторная работа №4. Решение задачи целочисленного программирования 150
6.5 Лабораторная работа №5. Решение задачи о коммивояжере 161
6.6 Лабораторная работа №6. Решение задачи безусловной оптимизации функций одной переменной 178
6.6.1 Метод равномерного поиска 178
6.6.2 Метод золотого сечения 180
6.6.3 Метод Ньютона 181
6.7 Лабораторная работа №7. Решение задачи безусловной оптимизации функций многих переменных 183
6.7.1 Метод циклического покоординатного спуска 183
6.7.2 Метод наискорейшего спуска 186
6.8 Лабораторная работа №8. Решение задачи нелинейного программирования методами Лагранжа и приведенного градиента Вулфа 191
6.8.1 Метод неопределенных множителей Лагранжа 191
6.8.2 Метод приведенного градиента Вулфа 200
6.9 Лабораторная работа №9. Решение задачи нелинейного программирования методом штрафных функций 214
Список использованной литературы 222

Введение

Математическое программирование представляет собой математическую дисциплину, занимающуюся изучением экстремальных задач и разработкой методов их решения.

В общем виде математическая постановка экстремальной задачи состоит в определении наибольшего или наименьшего значения целевой функции $f(x_1, x_2, \ldots, x_n)$ при условиях $g_i(x_1, x_2, \ldots, x_n) < b_i$ $(i = 1,m)$, где f и g_i – заданные функции, а b_i – некоторые действительные числа.

В зависимости от свойств функций f и g_i математическое программирование можно рассматривать как ряд самостоятельных дисциплин, занимающихся изучением и разработкой методов решения определенных классов задач.

Прежде всего, задачи математического программирования делятся на задачи линейного и нелинейного программирования. При этом если все функции f и g_i линейные, то соответствующая задача является *задачей линейного программирования.* Если же хотя бы одна из указанных функций нелинейная, то соответствующая задача является *задачей нелинейного программирования.*

Наиболее изученным разделом математического программирования является линейное программирование. Для решения задач линейного программирования разработан целый ряд эффективных методов, алгоритмов и программ.

Среди задач нелинейного программирования наиболее глубоко изучены *задачи выпуклого программирования.* Это задачи, в результате решения которых определяется минимум выпуклой (или максимум вогнутой) функции, заданной на выпуклом замкнутом множестве.

В свою очередь, среди задач выпуклого программирования более подробно исследованы *задачи квадратичного программирования.* В результате решения таких задач требуется в общем случае найти максимум (или минимум) квадратичной функции при условии, что ее переменные удовлетворяют некоторой системе линейных не-

равенств или линейных уравнений либо некоторой системе, содержащей как линейные неравенства, так и линейные уравнения.

Каждая глава данного учебного пособия содержит детально разобранные примеры и иллюстрации, а излагаемые методы доведены до алгоритмов их решения на ЭВМ.

Кроме того, шестая глава целиком посвящена комплексу лабораторных работ, включающих коды программ, описания входных и выходных параметров, решения контрольных примеров, а также варианты заданий.

Авторы благодарны рецензентам за их ценные замечания в ходе подготовки учебного пособия к изданию.

Линейное программирование

1.1 Примеры задач линейного программирования

1.1.1 Задача планирования выпуска продукции (планирование производства)

Машиностроительное предприятие для изготовления четырех видов продукции использует токарное, фрезерное, сверлильное, расточное и шлифовальное оборудование, а также комплектующие изделия. Кроме того, сборка изделий требует выполнения сборочно-наладочных работ. Нормы затрат всех видов ресурсов на изготовление каждого из изделий приведены в таблице (см. табл. 1.1). В этой же таблице указан имеющийся фонд каждого из ресурсов и прибыль от реализации одного изделия каждого вида.

Таблица 1.1–Нормы затрат на изготовление одного изделия

Ресурсы	Нормы затрат на изготовление одного изделия				Общий объем ресурсов
1	2	3	4	5	6
Производительность оборудования (чел.-час)					
Токарного	550	-	620	-	64270
Фрезерного	40	30	20	20	4800
Сверлильного	86	110	150	52	22360
Расточного	160	92	158	128	26240
Шлифовального	-	158	30	50	7900
Комплект изделия (шт.)	3	4	3	3	520
Сборочно-наладочные работы (чел.-час.)	4,5	4,5	4,5	4,5	720

Продолжение таблицы 1.1

Ресурсы	Нормы затрат на изготовление одного изделия				Общий объем ресурсов
1	2	3	4	5	6
Прибыль от реализации одного изделия (руб.)	315	278	573	370	

Надо составить такой план выпуска продукции, чтобы получить максимальную прибыль. Построим математическую модель данной задачи. Пусть x_1 – планируемое количество изделий 1-го типа, x_2, x_3, x_4 – планируемое количество изделий 2-го, 3-го, 4-го типов соответственно.

Тогда

$$f=315x_1+278x_2+573x_3+370x_4 \qquad (1.1)$$

прибыль предприятия. Так как нам надо получить как можно большую прибыль, то будем искать наибольшее (максимальное) значение функции f. Общие объемы ресурсов накладывают на нашу задачу ограничения

$$\begin{gathered} 550x_1 + 620x_2 \le 64270 \\ 40x_1 + 30x_2 + 20x_3 + 20x_4 \le 4800 \\ 86x_1 + 110x_2 + 150x_3 + 52x_4 \le 22360 \\ 160x_1 + 92x_2 + 158x_3 + 128x_4 \le 26240 \\ 158x_2 + 30x_3 + 50x_4 \le 7900 \\ 3x_1 + 4x_2 + 3x_3 + 3x_4 \le 520 \\ 4.5x_1 + 4.5x_2 + 4.5x_3 + 4.5x_4 \le 720. \end{gathered} \qquad (1.2)$$

Так как целевая функция и ограничения являются линейными, то это задача линейного программирования.

1.1.2 Задача планирования капитальных вложений

Для электроснабжения трех стройплощадок используются две трансформаторные подстанции. Первая стройплощадка потребляет 130 кВт, а вторая и третья по 140 кВт. Мощность первой ТП – 250 кВт, а второй 160 кВт. Расстояние от первой ТП – 600, 400, 200 м, до первой, второй и третьей площадки соответственно. От второй ТП: – 500, 300 и 200 м соответственно. Требуется составить схему электроснабжения с минимальными капитальными вложениями.

Капитальные вложения для устройства линий рассчитываются по следующей формуле:

$$\sum_{j=1}^{3}\sum_{i=1}^{2}\frac{S_{ij}K_0}{\sqrt{3U_j}}l_{ij} = \alpha\sum_{j=1}^{3}\sum_{i=1}^{2}S_{ij}l_{ij}\,, \qquad (1.3)$$

где: S_{ij} – мощность, потребляемая j-ой площадкой от i-ой ТП, кВт;

l_{ij} – расстояние от j-ой площадки до i-ой ТП, м.;

α – постоянный множитель.

Итак, нам надо минимизировать функцию

$$f=\alpha(6S_{11}+4S_{12}+2S_{13}+5S_{21}+3S_{22}+2S_{23}), \qquad (1.4)$$

при ограничениях:

$$\begin{aligned} S_{11} + S_{21} &= 130;\\ S_{12} + S_{22} &= 140;\\ S_{13} + S_{23} &= 140;\\ S_{11} + S_{12} + S_{13} &= 250;\\ S_{21} + S_{22} + S_{23} &= 160\,. \end{aligned} \qquad (1.5)$$

Как и в предыдущем примере имеем задачу линейного программирования.

1.2 Основные определения

ОПРЕДЕЛЕНИЕ 1.1. Задача, состоящая в нахождении наибольшего (наименьшего) значения функции

$$f(\boldsymbol{x})=c_1x_1+c_2x_2 + \ldots + c_{\mathrm{n}}x_{\mathrm{n}} \qquad (1.6)$$

на множестве точек $\boldsymbol{x}^T=(x_1,\ldots,x_n)$, удовлетворяющих системе ограничений вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 +\ldots+ a_{1n}x_n < R_1 > b_1 \\ a_{21}x_1 + a_{22}x_2 +\ldots+ a_{2n}x_n < R_2 > b_2 \\ . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ a_{m1}x_1 + a_{m2}x_2 +\ldots+ a_{mn}x_n < R_m > b_m \end{cases} \qquad (1.7)$$

называется задачей линейного программирования общего вида.

Здесь:

$f(\boldsymbol{x})=c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$ – целевая функция;

R_i, $i = \overline{1,m}$ – операции отношения =, ≥,≤;

c_i, $i = \overline{1,n}$; a_{ij}, $i = \overline{1,m}$, $j = \overline{1,n}$; b_i, i = $\overline{1,m}$ – заданные константы.

ОПРЕДЕЛЕНИЕ 1.2. Всякую точку $\boldsymbol{x}^T=(x_1,\ldots,x_n)$, компоненты которой удовлетворяют всем ограничениям системы (1.7), будем называть допустимой точкой или допустимым решением задачи, или допустимым планом задачи.

Задача линейного программирования состоит, таким образом, в нахождении такой допустимой точки $\boldsymbol{x}^{(0)}$ (такого допустимого плана) среди множества допустимых точек, при которой целевая функция принимает max (min) значение. Допустимое решение $\boldsymbol{x}^{(0)}= (x_1^{(0)},\ldots,x_n^{(0)})^T$, доставляющее целевой функции оптимальное значение (оптимум), будет называться оптимальным решением или оптимальным планом задачи. В дальнейшем будем говорить, к примеру, только о нахождении max целевой функции.

Задачу линейного программирования, представленную в виде:

$$
\begin{aligned}
& f = c_1x_1 + c_2x_2 + \ldots + c_nx_n \to \max\ (\min) \\
& \begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n = b_2 \\ . \quad . \quad . \quad . \quad . \quad . \quad . \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \end{cases} \\
& x_i \geq 0,\ i = \overline{1,n};
\end{aligned}
\tag{1.8}
$$

будем считать канонической задачей линейного программирования.

Введением дополнительных переменных, мы можем любое ограничение вида неравенства свести к ограничениям – равенствам. Если не на все переменные наложено требование неотрицательности (слабые ограничения), то такие переменные можно заменить разностью двух новых переменных, каждая из которых неотрицательна. Проиллюстрируем сказанное на примере.

Пример 1.1. Привести к каноническому виду задачу:

$$
\begin{aligned}
& f = x_1 - 2x_1 + x_3 \to \max \\
& \begin{cases} x_1 + 2x_2 + 3x_3 \leq 5 \\ 2x_1 + 3x_2 - 4x_3 = 3 \\ -x_1 + x_2 - x_3 \geq 2 \end{cases} \\
& x_1 \geq 0,\ x_2 \geq 0.
\end{aligned}
\tag{1.9}
$$

Введем две дополнительные неотрицательные переменные x_4 и x_5 в первом и третьем ограничениях, так, чтобы получить ограничения – равенства. Кроме того, переменную x_3 представим в виде:

$$x_3 = x_3' - x_3'', \tag{1.10}$$

где: $x_3' \geq 0$, $x_3'' \geq 0$.

Получим:

$$
\begin{aligned}
& f = x_1 - 2x_1 + x_3' - x_3'' \to \max, \\
& \begin{cases} x_1 + 2x_2 + 3(x_3' - x_3'') + x_4 = 5 \\ 2x_1 + 3x_2 - 4(x_3' - x_3'') = 3 \\ -x_1 - x_2 - (x_3' - x_3'') - x_5 = 2 \end{cases} \\
& x_1 \geq 0,\ x_2 \geq 0,\ x_3' \geq 0,\ x_3'' \geq 0,\ x_4\ 0,\ x_5 \geq 0,
\end{aligned}
\tag{1.11}
$$

а это уже задача в канонической форме.

Введя в рассмотрение векторы

$$\boldsymbol{c}=(c_1, c_2, \ldots c_n),\ \boldsymbol{x}=(x_1, \ldots, x_n)^T,\ \boldsymbol{b}=(b_1,\ldots, b_m)^T,\quad \boldsymbol{A}=\begin{pmatrix} a_{11} \ldots a_{1n} \\ a_{21} \ldots a_{2n} \\ \ldots\ldots\ldots \\ a_{m1} \ldots a_{mn} \end{pmatrix}$$

можем переписать задачу (1.6),(1.7) линейного программирования в форме:

$$f=\boldsymbol{c}\cdot\boldsymbol{x} \rightarrow \max(\min) \qquad (1.12)$$

$$\begin{cases} \boldsymbol{Ax}=\boldsymbol{b} \\ \boldsymbol{x}\geq 0. \end{cases} \qquad (1.13)$$

1.3 Геометрическая интерпретация двумерной задачи линейного программирования и ее решение

Рассмотрим двумерную задачу:

$$f = c_1x_1 + c_2x_2 \rightarrow \max(\min), \qquad (1.14)$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 < R_1 > b_1 \\ a_{21}x_1 + a_{22}x_2 < R_2 > b_2 \\ \ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 < R_m > b_m\ . \end{cases} \qquad (1.15)$$

Каждое из ограничений $a_{i1}x_1+a_{i2}x_2 < R_i > b_i$ определяет в плоскости, с системой координат x_1 о x_2 множество точек, лежащих по одну сторону от прямой $a_{i1}x_1+a_{i2}x_2=b_i$ (т.е. полуплоскость). Множество всех точек плоскости, координаты которых удовлетворяют всем ограничениям, т.е. принадлежат сразу всем полуплоскостям, определяемым отдельными ограничениями, будет представлять собой допустимое множество. Очевидно, что, если множество не пусто, то это будет некоторый многоугольник (возможно и неограниченный, возможно вырождающийся в отрезок, точку). Многоугольник будет выпуклым, что легко показать, т.е. любые две его точки можно соеди-

нить отрезком, точки которого принадлежат допустимому множеству.

ОПРЕДЕЛЕНИЕ 1.3. Множество D-точек n-мерного евклидова пространства будем называть выпуклым, если для любых $\boldsymbol{x}^{(1)}=(x_1^{(1)},\ldots,x_n^{(1)})^T$ и $\boldsymbol{x}^{(2)}=(x_1^{(2)},\ldots,x_n^{(2)})^T$ из множества D и любых $\alpha \geq 0$, $\beta \geq 0$ таких, что $\alpha+\beta=1$, точка $\boldsymbol{x}=\alpha\boldsymbol{x}^{(1)}+\beta\boldsymbol{x}^{(2)}$ также принадлежит D.

ОПРЕДЕЛЕНИЕ 1.4. Вершиной выпуклого множества в R_n назовем такую точку, которую нельзя представить в виде $\boldsymbol{x}=\alpha\boldsymbol{x}^{(1)}+\beta\boldsymbol{x}^{(2)}$, $\alpha>0$, $\beta>0$, $\alpha+\beta=1$, ни при каких $\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)}$.

Проиллюстрируем вышесказанное примерами.

Примеры: изобразить множество точек, удовлетворяющих системе неравенств.

1) $\begin{cases} 2x_1 - 3x_2 \leq 2 \\ 3x_1 - x_2 \leq 1 \\ -x_1 + \frac{1}{2}x_2 \leq 1 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$

2) $\begin{cases} -x_1 + x_2 \leq 1 \\ x_1 - 2x_2 \leq 1 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$

3) $\begin{cases} \frac{3}{4}x_1 + x_2 \geq 3 \\ x_1 + x_2 \leq 1 \\ x_2 \leq 1 \end{cases}$

4) $\begin{cases} 3x_2 - \frac{1}{2}x_1 \leq 3 \\ x_1 + x_2 \geq 1 \\ x_1 + x_2 \leq 1 \\ x_2 - x_1 \geq -1 \end{cases}$

Построив допустимые множества, убедимся в вышесказанном.

Рассмотрим теперь геометрическую интерпретацию решения задачи линейного программирования. Пусть допустимая область задачи (1.14)–(1.15) оказалась непустой (в соответствии с рисунком 1.1).

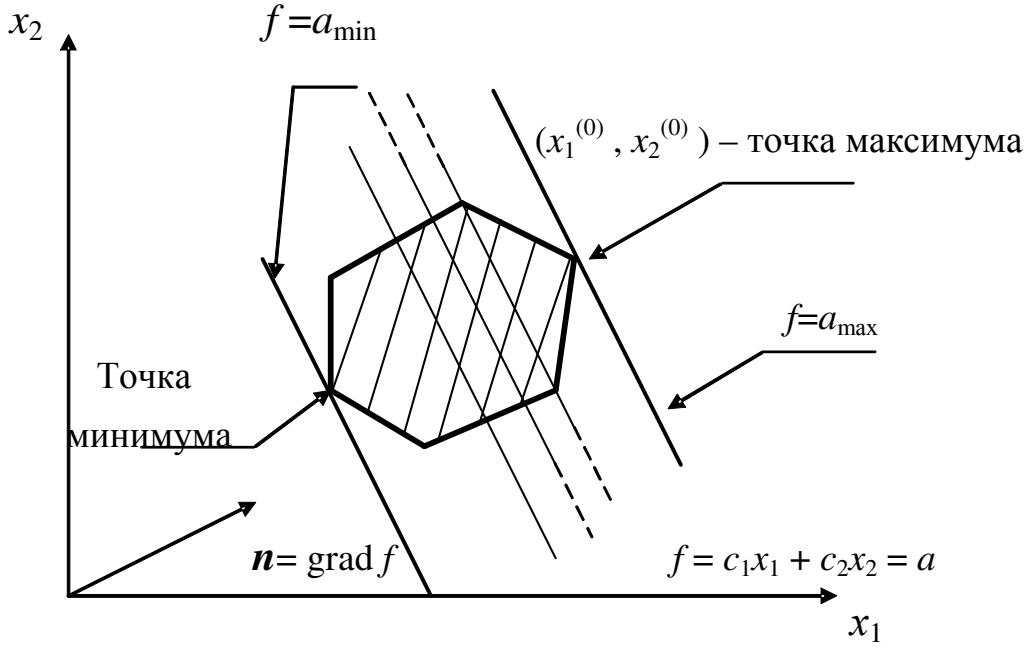


Рисунок 1.1 - Геометрическая интерпретация задачи линейного программирования

Мы хотим найти те точки допустимой области, координаты которых доставляют целевой функции наибольшее значение. Построим линию уровня целевой функции $f=c_1x_1+c_2x_2=a$. Получим множество точек, в точках которого f принимает одно и тоже значение a . Перемещая линию уровня в направлении вектора $grad\ f=(c_1,c_2)=\boldsymbol{n}$, нормального к линии уровня, будем получать в пересечении этой линии с допустимой областью точки, в которых целевая функция принимает новое значение, большее чем значение на предшествующих линиях уровня.

Пересечение допустимой области с линией уровня в том ее положении, когда дальнейшее перемещение дает пустое множество, и будет множеством точек максимума задачи линейного программирования. Перемещая линию уровня в направлении противоположном вектору $\boldsymbol{n}$, аналогичным образом найдем точки минимума.

Пример 1.2.

$$f = x_2 - 2x_1 \to \max$$
$$\begin{cases} -x_1 + x_2 \le 1 \\ x_1 - 2x_2 \le 1 \\ x_1 \ge 0,\ x_2 \ge 0 \end{cases}$$

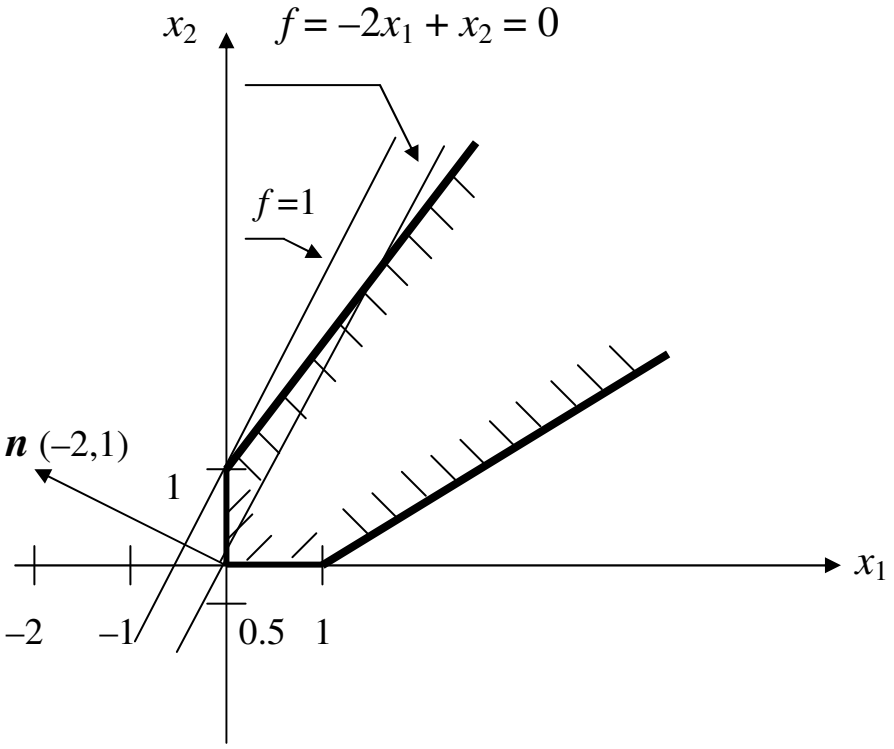


Рисунок 1.2 - Геометрическая интерпретация задачи линейного программирования

Точка максимума:

$$\boldsymbol{x}=(0,\ 1)^{\mathrm{T}};$$

$$f_{\max}=f(0,\ 1)=1.$$

Легко представить, что задача имеет бесчисленное множество оптимальных решений, а может их не иметь вовсе, как например:

$$f = x_1 + x_2 \rightarrow \min(\max);$$

$$\begin{cases} x_1 + x_2 \geq 1 \\ -x_1 + x_2 \leq 1 \\ x_1 - 2x_2 \leq 2 \end{cases}$$

$$x_1 \geq 0,\ x_2 \geq 0.$$

Целевая функция достигает минимального значения в точках отрезка $x_1+x_2=1$, между точками (0, 1) и (1, 0), а максимального значения функция не достигает (не ограничена в допустимой области).

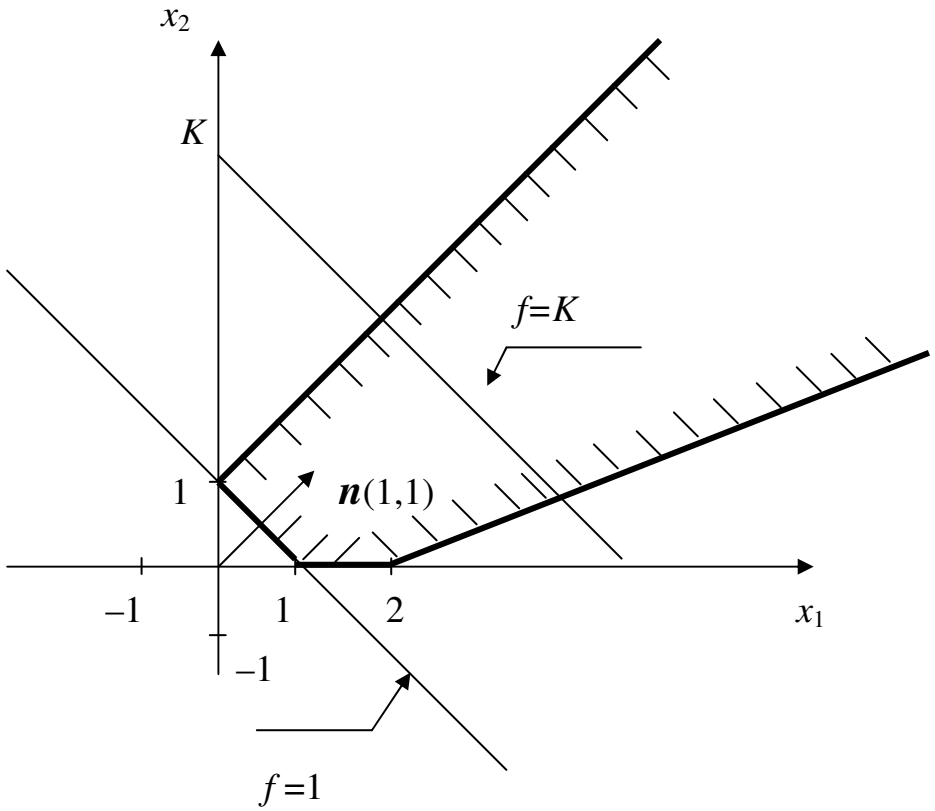


Рисунок 1.3 – Геометрическая интерпретация задачи линейного программирования

1.3.1 Свойства задачи линейного программирования

Рассмотренные примеры позволяют сформулировать основные свойства задачи линейного программирования.

Свойство 1. Допустимая область задачи линейного программирования выпукла, если она не пуста.

Свойство 2. Если допустимая область имеет вершины и задача линейного программирования имеет решение, то оно достигается по крайней мере в одной из вершин.

Свойство 3. Множество решений задачи линейного программирования выпукло.

Свойство 4. Если допустимая область ограничена, то любая задача линейного программирования имеет оптимальное решение.

Свойство 5. Необходимым и достаточным условием существования решения задачи линейного программирования на максимум (минимум) является ограниченность

целевой функции сверху (соответственно снизу) в допустимой области.

Все перечисленные свойства справедливы и в общем случае ($n \geq 2$).

Свойства задачи линейного программирования наталкивают на следующую схему решения задачи линейного программирования, известную как симплекс-метод. Именно: пусть рассматриваемая задача имеет непустое допустимое множество с вершинами.

Тогда:

1) тем или иным способом находим какую-нибудь вершину допустимого множества и по определенным критериям определяем, не является ли она оптимальной. Если вершины нет, то допустимая область пуста. Если вершина оптимальна, то задача решена.

Если нет, то

2) используя определенные правила проверяем, нельзя ли утверждать, что задача не имеет оптимального решения (целевая функция не ограничена сверху или, соответственно, снизу на допустимом множестве). Если утверждать это можно, то задача неразрешима.

Если нельзя, то

3) по определенному правилу ищем новую, лучшую вершину и переходим к пункту 1).

Итак, для реализации предложенной схемы необходимо указать способ нахождения вершины допустимого множества, критерий оптимальности или неразрешимости, способ перехода от одной вершины к другой, лучшей.

1.3.2 Обоснование симплекс метода

Пусть имеется задача линейного программирования в канонической форме:

$$f = c_1x_1 + c_2x_2 \ \ldots \ c_nx_n \to \max \tag{1.16}$$

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \ldots\ldots\ldots\ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \\ x_1 \geq 0, \ldots, x_m \geq 0 . \end{cases} \tag{1.17}$$

Вводя в рассмотрение векторы

$$\boldsymbol{A}_j=(a_{1j}, a_{2j}, \ldots, a_{mj})^T, \ j=\overline{1,n}; \tag{1.18}$$

мы можем переписать задачу в форме

$$\begin{aligned} &f = (\boldsymbol{c}, \boldsymbol{x}) \to \max; \\ &\boldsymbol{A}_1x_1 + \boldsymbol{A}_2x_2 + \ldots + \boldsymbol{A}_nx_n = \boldsymbol{b}; \\ &\boldsymbol{x} \geq 0; \end{aligned} \tag{1.19}$$

и трактовать ее следующим образом: из всех разложений вектора $\boldsymbol{b}$ по векторам $\boldsymbol{A}_1,\ldots,\boldsymbol{A}_n$, с неотрицательными коэффициентами требуется выбрать хотя бы одно такое, коэффициенты x_i, $i=1,n$, которого доставляют целевой функции f оптимальное значение. Не ограничивая общности считаем ранг матрицы $\boldsymbol{A}$ равным m и $n>m$ (случай $n=m$ тривиален). Дадим ряд определений.

ОПРЕДЕЛЕНИЕ 1.5. Ненулевое допустимое решение $\boldsymbol{x}=(x_1,\ldots,x_n)^T$ называется опорным, если векторы $\boldsymbol{A}_j$, соответствующие отличным от нуля координатам вектора $\boldsymbol{x}$, линейно-независимы.

ОПРЕДЕЛЕНИЕ 1.6. Ненулевое опорное решение назовем невырожденным, если оно имеет точно «m» положительных координат.

ОПРЕДЕЛЕНИЕ 1.7. Если число положительных координат опорного решения меньше m, то оно называется вырожденным.

ОПРЕДЕЛЕНИЕ 1.7. Упорядоченный набор из «m» линейно – независимых векторов $\boldsymbol{A}_i$, соответствующих положительным координатам опорного решения назовем базисом.

Пример 1.3.

Дана система ограничений задачи:

$$\begin{cases} 2x_1 + x_2 + 3x_3 = 2 \\ x_1 - 2x_3 + x_4 = 1 \\ x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0. \end{cases} \qquad (1.20)$$

Здесь

$$m=2;\ \boldsymbol{A}_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix},\ \boldsymbol{A}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ \boldsymbol{A}_3 = \begin{pmatrix} 3 \\ -2 \end{pmatrix},\ \boldsymbol{A}_4 = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ \boldsymbol{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Очевидно, что $\boldsymbol{x}^{(1)}=(0, 2, 0, 1)^T$, $\boldsymbol{x}^{(2)}=(1, 0, 0, 0)^T$, $\boldsymbol{x}^{(3)}=(\frac{1}{2}, 1, 0, \frac{1}{2})^T$, являются допустимыми решениями задачи. Очевидно, что $\boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(2)}$– являются опорными решениями, поскольку системы векторов { $\boldsymbol{A}_2, \boldsymbol{A}_4$} и { $\boldsymbol{A}_3$}, линейно независимы, а $\boldsymbol{x}^{(3)}$ не является опорным решением, так как { $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_4$ } – линейно зависимы. Опорное решение $\boldsymbol{x}^{(1)}$– невырожденное, а $\boldsymbol{x}^{(2)}$– вырожденное. Базис невырожденного опорного решения определяется естественным образом. Для $\boldsymbol{x}^{(1)}$ – это { $\boldsymbol{A}_2, \boldsymbol{A}_4$}.

Приведем теорему, увязывающую понятие опорного решения и вершины допустимого множества.

ТЕОРЕМА 1.1. Вектор $\boldsymbol{x}= (x_1,\ldots,x_n)$ тогда и только тогда является опорным решением задачи, когда точка $\boldsymbol{x}$ является вершиной допустимого множества.

Доказательство. Необходимость. Пусть $\boldsymbol{x}$ – опорное решение задачи (1.19). Если $\boldsymbol{x}$ = 0, то невозможность ее представления в виде $\boldsymbol{x} = \alpha\boldsymbol{x}^{(1)} + \beta\boldsymbol{x}^{(2)}$, где $\alpha > 0$, $\beta > 0$ и $\boldsymbol{x}^{(1)} \neq \boldsymbol{x}^{(2)}$ – допустимые решения, очевидна.

Пусть $\boldsymbol{x} \neq 0$ и

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_k, \underbrace{0,\ldots,0}_{n-k}),$$

где $x_1>0$, $x_2 >0,\ldots,$ $x_k>0$.

Предположим, в противоречие с теоремой, что точка $\boldsymbol{x}$ не является вершиной допустимого многогранника:

$$\boldsymbol{x}^{(1)}=(x_1^{(1)}, x_2^{(1)},..., x_n^{(1)}) \text{ и } \boldsymbol{x}^{(2)}=(x_1^{(2)}, x_2^{(2)},..., x_n^{(2)}),$$

$$\boldsymbol{x}=\alpha\boldsymbol{x}^{(1)}+\beta\boldsymbol{x}^{(2)}, \alpha>0, \beta>0, \alpha+\beta=1.$$

Так как последние n-k координат точки $\boldsymbol{x}$ равны нулю, то и последние n-k координат, как точки $\boldsymbol{x}^{(1)}$, так и точки $\boldsymbol{x}^{(2)}$ должны быть равны 0.Таким образом, можно записать:

$$\boldsymbol{x}^{(1)}=(x_1^{(1)}, x_2^{(1)},..., x_k^{(1)},0,...,0),$$

$$\boldsymbol{x}^{(2)}=(x_1^{(2)}, x_2^{(2)},..., x_k^{(2)},0,...,0).$$

Ввиду допустимости точек $\boldsymbol{x}^{(1)}$ и $\boldsymbol{x}^{(2)}$ имеем:

$$\boldsymbol{b}=x_1^{(1)}\boldsymbol{A}_1+x_2^{(1)}\boldsymbol{A}_2+...+x_k^{(1)}\boldsymbol{A}_k \qquad (1.21)$$

$$\boldsymbol{b}=x_1^{(2)}\boldsymbol{A}_1+x_2^{(2)}\boldsymbol{A}_2+...+x_k^{(2)}\boldsymbol{A}_k. \qquad (1.22)$$

Вычитая почленно (1.22) из (1.21), получим:

$$(x_1^{(1)}-x_1^{(2)})\boldsymbol{A}_1+(x_2^{(1)}-x_2^{(2)})\boldsymbol{A}_2+...+(x_k^{(1)}-x_k^{(2)})\boldsymbol{A}_k=0. \qquad (1.23)$$

Точки $\boldsymbol{x}^{(1)}$ и $\boldsymbol{x}^{(2)}$ различны, следовательно, среди коэффициентов при $\boldsymbol{A}_i$ (i=1,2,…,k) в (1.23) есть отличные от нуля. А это означает, что векторы $\boldsymbol{A}_1, \boldsymbol{A}_2,…, \boldsymbol{A}_k$ линейно зависимы, что противоречит тому, что $\boldsymbol{x}=(x_1,x_2,…,x_k,0,…,0)$– опорное решение задачи. Следовательно, наше предположение неверно и $\boldsymbol{x}$ вершина допустимого многогранника.

Достаточность. Пусть наоборот, точка $\boldsymbol{x}$– вершина допустимого многогранника. Если $\boldsymbol{x}$ = 0, то $\boldsymbol{x}$ по условию опорное решение. Пусть, $\boldsymbol{x}$ ≠0 и для определенности $\boldsymbol{x}=(x_1,x_2,…,x_k,0,…,0)$, где $x_1>0$, $x_2>0,…,x_k>0$.

Предположим снова от противного, что при этом вектор $\boldsymbol{x}=(x_1,x_2,…,x_k,0,…,0)$ являясь, конечно, допустимым для нашей задачи, не является опорным решением. Тогда векторы $\boldsymbol{A}_1, \boldsymbol{A}_2,…, \boldsymbol{A}_k$ – линейно зависимы, т.е. существуют такие действительные числа $\alpha_1, \alpha_2,…, \alpha_k$ не все равные нулю, что справедливо равенство:

$$\alpha_1\boldsymbol{A}_1+\alpha_2\boldsymbol{A}_2+...+\alpha_k\boldsymbol{A}_k=0. \qquad (1.24)$$

Являясь допустимым, вектор $\boldsymbol{x}$ удовлетворяет условию:

$$x_1\boldsymbol{A}_1 + x_2\boldsymbol{A}_2 + ... + x_k\boldsymbol{A}_k = \boldsymbol{b} \ . \qquad (1.25)$$

Умножим обе части равенства (1.24) на некоторый параметр θ и сначала вычтем почленно (1.24) из (1.25), а затем сложим (1.24) и (1.25).Получим:

$$(x_1 - \theta\alpha_1)\boldsymbol{A}_1 + (x_2 - \theta\alpha_2)\boldsymbol{A}_2 + \ldots + (x_k - \theta\alpha_k)\boldsymbol{A}_k = \boldsymbol{b},$$

$$(x_1 + \theta\alpha_1)\boldsymbol{A}_1 + (x_2 + \theta\alpha_2)\boldsymbol{A}_2 + \ldots + (x_k + \theta\alpha_k)\boldsymbol{A}_k = \boldsymbol{b}.$$

Так как $\boldsymbol{x}_i > 0$ (i=1,2,…,k), то, очевидно, можно найти такое достаточно малое значение $\theta_0 > 0$ множителя θ, что и в первом и во втором из полученных равенств, все коэффициенты будут неотрицательными. Тогда эти равенства (при $\theta = \theta_0$) означают, что *n*- мерные векторы

$$\boldsymbol{x}^{(1)} = (x_1 - \theta_0\alpha_1; x_2 - \theta_0\alpha_2; \ldots; x_k - \theta_0\alpha_k; 0; \ldots; 0)$$

и

$$\boldsymbol{x}^{(1)} = (x_1 + \theta_0\alpha_1; x_2 + \theta_0\alpha_2; \ldots; x_k + \theta_0\alpha_k; 0; \ldots; 0)$$

являются допустимыми (причем различными) решениями задачи.

При этом, очевидно, имеем:

$$\frac{1}{2}\boldsymbol{x}^{(1)} + \frac{1}{2}\boldsymbol{x}^{(2)} = (x_1, x_2, \ldots, x_k, 0, \ldots, 0) = \boldsymbol{x},$$

а это противоречит тому, что $\boldsymbol{x}$– вершина допустимого многогранника. Следовательно, $\boldsymbol{x}$ не может не быть опорным решением. Теорема доказана.

Таким образом, задача о нахождении вершины допустимого множества свелась к задаче нахождения опорного решения, а, следовательно, к нахождению базиса. Будем считать, что исходный базис $\boldsymbol{A}_{i_1}, \boldsymbol{A}_{i_2}, \ldots, \boldsymbol{A}_{i_m}$ дан. Отправляясь от него покажем, как найти опорное решение. Сформулируем условие оптимальности решения, условие отсутствия решения. Покажем, как перейти к базису, дающему лучшее решение.

Дан базис

$$\boldsymbol{A}_{i_1}, \ \boldsymbol{A}_{i_2}, \ldots, \ \boldsymbol{A}_{i_m}.$$

Тогда:

$$\boldsymbol{b}=x_{i_{10}}\boldsymbol{A}_{i_1}+x_{i_{20}}\boldsymbol{A}_{i_2}+\ldots+x_{i_{m0}}\boldsymbol{A}_{i_m}=[\boldsymbol{A}_{i_1},\ldots,\boldsymbol{A}_{i_m}]*(x_{i_{10}},\ldots,x_{i_{m0}})^{T},$$

$$(x_{i_{10}},\ldots,x_{i_{m0}})^{T}=[\boldsymbol{A}_{i_1},\ldots,\boldsymbol{A}_{i_m}]^{-1}*(\boldsymbol{b}_1,\ldots,\boldsymbol{b}_m)^{T},$$

дополнив найденный вектор нулевыми значениями переменных ($x_{i_{m+1}},\ldots,x_{i_n}$), получаем опорное решение. Исследуем его на оптимальность, для этого исследуем целевую функцию.

Предварительно введем обозначения:

$\boldsymbol{x}_B=(x_{i_1},\ldots,x_{i_m})^{T}$ – вектор базисных переменных;

$\boldsymbol{B}=[\boldsymbol{A}_{i_1},\ldots,\boldsymbol{A}_{i_m}]$– матрица базисных векторов;

$\boldsymbol{x}_D=(x_{i_{m+1}},\ldots,x_{i_n})$– вектор свободных переменных;

$\boldsymbol{c}_B=(c_{i_1},\ldots,c_{i_m})^{T}$ – вектор из коэффициентов целевой функции при базисных неизвестных;

$\boldsymbol{c}_D=(c_{i_{m+1}},\ldots,c_{i_n})^{T}$ – вектор из коэффициентов целевой функции при свободных переменных;

$\boldsymbol{D}=[\boldsymbol{A}_{i_{m+1}},\ldots,\boldsymbol{A}_{i_n}]$ – матрица свободных векторов (векторов, не вошедших в базис).

С учетом введенных обозначений ограничения (1.19) можно переписать в виде:

$$\boldsymbol{B}\boldsymbol{x}_B+\boldsymbol{D}\boldsymbol{x}_D=\boldsymbol{b}. \qquad (1.26)$$

Выразим базисные переменные через свободные

$$\boldsymbol{x}_B=\boldsymbol{B}^{-1}(\boldsymbol{b}-\boldsymbol{D}\boldsymbol{x}_D)=\boldsymbol{B}^{-1}\boldsymbol{b}-\boldsymbol{B}^{-1}\boldsymbol{D}\boldsymbol{x}_D, \qquad (1.27)$$

(при $\boldsymbol{x}_D=0$ получаем базисные компоненты опорного решения).

Подставим $\boldsymbol{x}_B$ в целевую функцию:

$$f=\boldsymbol{c}_B^{T}\boldsymbol{x}_B+\boldsymbol{c}_D^{T}\boldsymbol{x}_D=\boldsymbol{c}_B^{T}\boldsymbol{B}^{-1}\boldsymbol{b}-(\boldsymbol{c}_B^{T}\boldsymbol{B}^{-1}\boldsymbol{D}-\boldsymbol{c}_D^{T})\boldsymbol{x}_D. \qquad (1.28)$$

Обозначим:

$$\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_{n-m})=(\boldsymbol{c}_B^{T}\boldsymbol{B}^{-1}\boldsymbol{D}-\boldsymbol{c}_D^{T}) \qquad (1.29)$$

и назовем его вектором оценок.

Очевидно, что

$$\boldsymbol{\alpha}\,\boldsymbol{x}_c=\sum_{j=1}^{n-m}\alpha_j\cdot x_{i_{m+j}}. \qquad (1.30)$$

Отметим, что произведение $\boldsymbol{B}^{-1}\boldsymbol{A}_{\mathrm{j}}$ представляет разложение вектора $\boldsymbol{A}_{\mathrm{j}}$ по базису, следовательно, в столбцах матрицы $\boldsymbol{B}^{-1}\boldsymbol{D}$ стоят разложения векторов, не попавших в базис (свободных) по базису. Проанализируем выражение для целевой функции (1.28):

а) если все компоненты вектора оценок неотрицательные, то максимальное значение целевой функции достигается при $\boldsymbol{x}_B=\boldsymbol{B}^{-1}\boldsymbol{b}$ и $\boldsymbol{x}_D=0$, то есть построенное опорное решение – оптимальный план (решение) исходной задачи;

б) если среди компонент вектора оценок $\boldsymbol{\alpha}$ есть отрицательные, то опорное решение не оптимально, поскольку, как видно из (1.28)–(1.30), возможно увеличить значение f за счет некоторых компонент вектора $\boldsymbol{x}_D$; при этом возможны две ситуации:

1) отрицательны компоненты вектора оценок, к примеру, $\alpha_{\mathrm{k}1}, \alpha_{\mathrm{k}2},\ldots,\alpha_{\mathrm{k}s}$ и при этом, по крайней мере у одного из свободных векторов $\boldsymbol{A}_{i_{m+k1}}, \boldsymbol{A}_{i_{m+k2}},\ldots, \boldsymbol{A}_{i_{m+ks}}$ все компоненты разложения по базису не положительны, тогда из (1.27) видно, что можно неограниченно увеличить компоненты вектора $\boldsymbol{x}_B$ за счет увеличения компоненты вектора $\boldsymbol{x}_D$, а, следовательно, целевая функция не ограничена;

2) в случае положительных компонент разложения по базису свободных векторов существует лучшее опорное решение, которое можно получить, введя в базис свободный вектор $\boldsymbol{A}_{\mathrm{S}}$, соответствующий наименьшей из отрицательных компонент вектора оценок (с целью максимально прирастить f) и, выведя из базиса тот вектор $\boldsymbol{A}_{\mathrm{i_r}}$, исходного базиса, для которого:

$$\min_{x_{\mathrm{i_k s}}>0} \frac{x_{\mathrm{i_k}\,0}}{x_{\mathrm{i_k s}}} = \frac{x_{\mathrm{i_r}\,0}}{x_{\mathrm{i_r s}}} \qquad (1.31)$$

где: $x_{\mathrm{i_k}\,0}$ – компоненты $\boldsymbol{x}_B$; $x_{\mathrm{i_k S}}$– компоненты вектора $\boldsymbol{A}_{\mathrm{S}}$, вводимого в базис, $\mathrm{k}=\overline{1,m}$.

Заменив базис, мы можем вновь перейти к построению опорного решения и его исследования.

Отметим, что координаты всех векторов в новом базисе могут быть найдены через координаты векторов в старом базисе по формулам:

$$x'_{kj} = \begin{cases} \dfrac{x_{kj}}{x_{ks}}, & \text{при } k = r \\ x_{kj} - \dfrac{x_{rj}}{x_{rs}} x_{ks}, & j = \overline{0, n},\ \ k = \overline{1, m},\ \ k \neq r. \end{cases} \qquad (1.32)$$

В итоге алгоритм решения задачи линейного программирования можно изобразить схемой, представленной на рисунке 1.4.

1.3.3 Нахождение начального базиса.

В заключении обратимся к вопросу построения начального базиса. Здесь отметим, что, если в исходной задаче все ограничения имели вид “≤”, то в качестве начального базиса (как легко показать) можно брать базис из векторов при дополнительных переменных. В общем случае для построения начального базиса используется специальный прием, известный как метод искусственного базиса, изложенный ниже. Итак, пусть дана задача линейного программирования в канонической форме:

$$\begin{aligned} & f = \boldsymbol{c} \cdot \boldsymbol{x} \ \rightarrow \max; \\ & \begin{cases} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \boldsymbol{x} \geq 0. \end{cases} \end{aligned} \qquad (1.33)$$

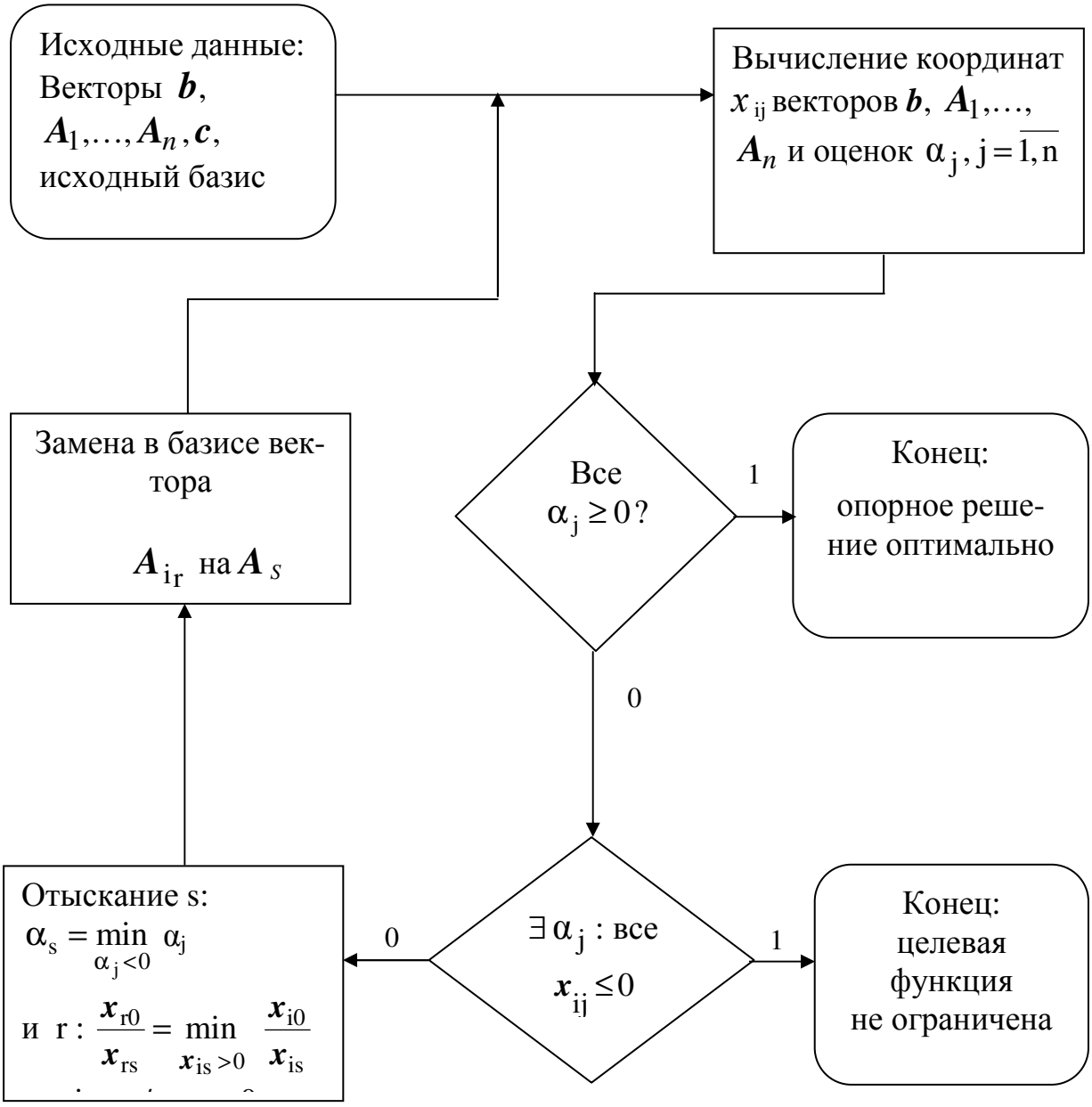


Рисунок 1.4 – Схема алгоритма решения задачи линейного программирования

Рассмотрим вспомогательную задачу:

$$G = -y_1 - y_2 - \ldots - y_m \to \max;$$

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n + y_1 = b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n + 0\cdot y_1 + y_2 = b_2 \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ a_{m1}x_1 + \ldots + a_{mn}x_n + 0*y_1 + \ldots + y_m = b_m \end{cases} \qquad (1.34)$$

$$x_i \geq 0,\ \ i = \overline{1,n},$$

$$y_i \geq 0,\ i = \overline{1,m}.$$

Так как целевая функция этой задачи ограничена сверху нулем, то задача имеет оптимальное решение.

Переменные $y_1, y_2, \ldots, y_m$ называются искусственными переменными.

Очевидно, что векторы

$$\boldsymbol{A}_{n+1}=(1,0,\ldots,0)^T,\ \boldsymbol{A}_{n+2}=(0,1,0,\ldots,0)^T,\ldots,\ \boldsymbol{A}_{n+m}=(0,0,\ldots,1)^T$$

образуют базис для опорного решения

$$x = (\underbrace{0,0,...,0}_{n},\ b_1, b_2, ..., b_m),$$

который называют искусственным базисом. Решая вспомогательную задачу симплекс-методом, мы найдем оптимальное решение $\boldsymbol{x}^{(0)}=(x_1^{(0)},\ldots,x_n^{(0)},y_1^{(0)},\ldots,y_m^{(0)})$. Если в этом решении, среди искусственных переменных есть положительные, то исходная задача линейного программирования неразрешима, если же $y_i^{(0)} = 0,\ i = \overline{1,m}$, то базис, соответствующий оптимальному решению вспомогательной задачи, можно взять в качестве исходного базиса основной задачи.

1.3.4 Решение в форме симплекс – таблиц.

В случае небольшого числа ограничений и переменных, приведенный в предыдущем разделе алгоритм можно легко реализовать без помощи ЭВМ, оформляя решение в виде симплекс-таблиц. Итак, дана задача:

$$\begin{aligned} & f = (\boldsymbol{c}, \boldsymbol{x}) \rightarrow \max \\ & \boldsymbol{A}_1 x_1 + \boldsymbol{A}_2 x_2 + \ldots + \boldsymbol{A}_n x_n = \boldsymbol{b}, \\ & \boldsymbol{x} \geq 0 \end{aligned} \qquad (1.35)$$

и найден исходный базис $\boldsymbol{A}_{i_1}, \boldsymbol{A}_{i_2}, \ldots, \boldsymbol{A}_{i_m}$. Находим разложения всех векторов $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n$ и $\boldsymbol{b}$ по базису. Полученные результаты заносим в таблицу 1.2.

Таблица 1.2 - Симплекс таблица

№ п/п	Базис	$\boldsymbol{c}_{B.}$	$\boldsymbol{b}$ (опорное решение)	c_1	c_2	…	c_n
				$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	…	$\boldsymbol{A}_n$
1	$\boldsymbol{A}_{i_1}$	c_{i_1}	x_{10}	x_{11}	x_{12}	…	x_{1n}
2	$\boldsymbol{A}_{i_2}$	c_{i_2}	x_{20}	x_{21}	x_{22}	…	x_{2n}
3	$\boldsymbol{A}_{i_3}$	c_{i_3}	x_{30}	x_{31}	x_{32}	…	x_{3n}
.	.	.	.	.	.	.	.
m	$\boldsymbol{A}_{i_m}$	c_{i_m}	x_{m0}	x_{m1}	x_{m2}	…	x_{mn}
Оценки		$f = \sum_{k=1}^{m} c_{i_k} x_{k0}$		α_1	α_2	…	α_n

Здесь в столбце «Базис» указываются вектора $\boldsymbol{A}_{i_k}$ попавшие в базис; в столбце $\boldsymbol{c}_{B.}$ записываются коэффициенты c_{i_k} при соответствующих неизвестных из целевой функции; в столбце $\boldsymbol{b}$ (опорное решение) записываются координаты разложения вектора $\boldsymbol{b}$ по базису, то есть опорный план; в столбцах $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n$ записывают координаты разложения этих векторов по базису, то есть столбцы матрицы $\boldsymbol{B}^{-1}\boldsymbol{D}$. После этого заполняют строку «Оценки»:

$f = \sum_{k=1}^{m} c_{i_k} x_{k0}$ – значение целевой функции для найденного опорного решения и оценки

$$\alpha_j = z_j - c_j = \sum_{k=1}^{m} c_{i_k} x_{kj} - c_j - \text{ компоненты вектора}$$

$$\boldsymbol{\alpha} = \boldsymbol{c}_B^{\mathrm{T}} \boldsymbol{B}^{-1} \boldsymbol{A}_j - c_j .$$

Далее, согласно алгоритму, выясняем, будет ли найденное опорное решение оптимальным, если да, то это решение (план) в столбце «$\boldsymbol{b}$», а если нет, то выбираем вектор подлежащий вводу в базис, вектор подлежащий выводу из базиса и переходим к нахождению следующего опорного решения, заполняя новую таблицу.

Пример 1.4.

$$\begin{cases} f = 5x_1 + 10x_2 \to \max \\ 4x_1 + 3x_2 \le 96 \\ 5x_1 + 8x_2 \le 144 \\ x_1 + 4x_2 \le 48 \\ x_1 \ge 0, \;\; x_2 \ge 0 . \end{cases}$$

Приведем задачу к каноническому виду:

$$\begin{cases} f = 5x_1 + 10x_2 + 0 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_5 \to \max \\ 4x_1 + 3x_2 + x_3 = 96 \\ 5x_1 + 8x_2 + x_4 = 144 \\ x_1 + 4x_2 + x_5 = 48 \\ x_i \ge 0, \;\; i = \overline{1,5} . \end{cases}$$

Введя векторы:

$$\boldsymbol{b} = \begin{pmatrix} 96 \\ 144 \\ 48 \end{pmatrix}, \; \boldsymbol{A}_1 = \begin{pmatrix} 4 \\ 5 \\ 1 \end{pmatrix}, \; \boldsymbol{A}_2 = \begin{pmatrix} 3 \\ 8 \\ 4 \end{pmatrix}, \; \boldsymbol{A}_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \; \boldsymbol{A}_4 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \; \boldsymbol{A}_5 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

можем записать задачу в виде:

$$\begin{cases} f = 5x_1 + 10x_2 + 0 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_5 \to \max \\ x_1 \boldsymbol{A}_1 + x_2 \boldsymbol{A}_2 + x_3 \boldsymbol{A}_3 + x_4 \boldsymbol{A}_4 + x_5 \boldsymbol{A}_5 = \boldsymbol{b} \end{cases}$$

$$x_i \geq 0, \quad i = \overline{1,5}.$$

Очевидно, что векторы $\boldsymbol{A}_3$, $\boldsymbol{A}_4$, $\boldsymbol{A}_5$ образуют базис и столь же очевидно, что относительно этого базиса:

$$\boldsymbol{b} = 96\boldsymbol{A}_3 + 144\boldsymbol{A}_4 + 48\boldsymbol{A}_5$$
$$\boldsymbol{A}_1 = 4\boldsymbol{A}_3 + 5\boldsymbol{A}_4 + 1\boldsymbol{A}_5$$
$$\boldsymbol{A}_2 = 3\boldsymbol{A}_3 + 8\boldsymbol{A}_4 + 4\boldsymbol{A}_5$$
$$\boldsymbol{A}_3 = 1\boldsymbol{A}_3 + 0\boldsymbol{A}_4 + 0\boldsymbol{A}_5$$
$$\boldsymbol{A}_4 = 0\boldsymbol{A}_3 + 1\boldsymbol{A}_4 + 0\boldsymbol{A}_5$$
$$\boldsymbol{A}_5 = 0\boldsymbol{A}_3 + 0\boldsymbol{A}_4 + 1\boldsymbol{A}_5$$

Заполняем первую симплекс таблицу 1.3:

Таблица 1.3 - Симплекс таблица

№ п/п	Базис	$\boldsymbol{c}_B$	$\boldsymbol{b}$ (опорное решение)	5	10	0	0	0
				$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	$\boldsymbol{A}_3$	$\boldsymbol{A}_4$	$\boldsymbol{A}_5$
1	$\boldsymbol{A}_3$	0	96	4	3	1	0	0
2	$\boldsymbol{A}_4$	0	144	5	8	0	1	0
3	$\boldsymbol{A}_5$	0	48	1	4	0	0	1
Оценки			f=0	α_1=-5	α_2=-10	α_3=0	α_4=0	α_5=0

Поскольку среди α_j есть отрицательные, то план не оптимален. Находим $\min\limits_{j=1,5} \alpha_j = \alpha_2 = -10$, следовательно, в новый базис будем вводить вектор $\boldsymbol{A}_2$ и столбец $\boldsymbol{A}_2$ назовем ведущим.

Найдем отношение координат вектора $\boldsymbol{b}$ к соответствующим, но только положительным координатам вектора $\boldsymbol{A}_2$ и выберем минимальное из них:

$$\min (96/3, 144/8, 48/4) = 48/4.$$

Из этого следует, что из базиса надо вывести $\boldsymbol{A}_5$. Строку «$\boldsymbol{A}_5$» будем называть ведущей. Число **4**, стоящее на пересечении ведущей строки и ведущего столбца, будем называть ведущим элементом.

Переходим к нахождению нового опорного решения, для чего составляем новую симплекс таблицу с новым базисом $\boldsymbol{A}_3$, $\boldsymbol{A}_4$, $\boldsymbol{A}_2$. Заполнение новой симплекс таблицы начинаем с заполнения строки, соответствующей вновь вводимому вектору $\boldsymbol{A}_2$. Она получается делением элементов ведущей строки на ведущий элемент. Остальные элементы таблицы можно найти исходя из формулы

$$x'_{ij} = x_{ij} - \frac{x_{rj}}{x_{rs}} x_{is}, \qquad (1.36)$$

где: x'_{ij} – искомый элемент в новой таблице;

x_{rs} – ведущий элемент;

x_{rj} – элемент, стоящий в проекции элемента x_{ij} на ведущий столбец;

x_{is} – элемент, стоящий в проекции x_{ij} на ведущую строку.

Таблица 1.4 Вторая симплекс таблица

№ п/п	Базис	$\boldsymbol{c}_B$	$\boldsymbol{b}$ (опорное решение)	5	10	0	0	0
				$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	$\boldsymbol{A}_3$	$\boldsymbol{A}_4$	$\boldsymbol{A}_5$
1	$\boldsymbol{A}_3$	0	60	13/14	0	1	0	-3/4
2	$\boldsymbol{A}_4$	0	48	3	0	0	1	-2
3	$\boldsymbol{A}_2$	10	12	1/4	1	0	0	1/4
Оценки			f=120	α_1=-5/2	0	0	0	5/2

Так как $\min \alpha_j = \alpha_1 = -5/2 < 0$, то план не оптимален и в базис следует ввести $\boldsymbol{A}_1$.

Найдем:

$$\min\left(\frac{60}{13/4}, \frac{48}{3}, \frac{12}{1/4}\right) = 48/3,$$

то есть из базиса выводим $\boldsymbol{A}_4$. Составляем симплекс таблицу (таблица 1.5).

Таблица 1.5 Третья симплекс таблица

№ п/п	Базис	$\boldsymbol{c}_B$	$\boldsymbol{b}$ (опорное решение)	5	10	0	0	0
				$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	$\boldsymbol{A}_3$	$\boldsymbol{A}_4$	$\boldsymbol{A}_5$
1	$\boldsymbol{A}_3$	0	8	0	0	1	-13/12	17/12
2	$\boldsymbol{A}_1$	5	16	1	0	0	1/3	-2/3
3	$\boldsymbol{A}_2$	10	8	0	1	0	-1/12	5/12
Оценки			f=160	0	0	0	5/6	5/6

Так как все $\alpha_j \geq 0$, то найденное опорное решение x_1=16, x_2 =8, x_3 =8, x_4=0, x_5=0 доставляет целевой функции f максимальное значение

$$f_{\max} = 5 \cdot 16 + 10 \cdot 8 = 160.$$

1.4 Двойственная задача линейного программирования

1.4.1 Пример прямой и двойственной задачи линейного программирования

Обратимся к задаче рационального использования ресурсов. Пусть для осуществления l технологических процессов $T_1,\ldots,T_l$ предприятию требуется m ресурсов $S_1,\ldots, S_m$. Запасы ресурсов каждого вида ограничены и равны $b_1,\ldots,b_m$.

Известен расход ресурсов $\{a_{ij}\}$, $i=\overline{1,m}$, $j=\overline{1,l}$ на единицу продукции по каждому технологическому процессу и стоимость единицы продукции каждого типа c_j, $j=\overline{1,l}$. Тогда, как нам известно, задача определения такого плана выпуска продукции x_j, при котором доход от ее реализации будет максимальным, является задачей линейного программирования:

$$\sum_{j=1}^{l} c_j x_j \to \max; \qquad (1.37)$$

$$\sum_{j=1}^{l} a_{ij} x_j \le b_i, i=\overline{1,m}; \qquad (1.38)$$

$$x_j \ge 0, \quad j = \overline{1,l}$$

и ее решение может быть найдено симплекс-методом.

Зададимся вопросом, какова с точки зрения предприятия ценность имеющихся в его распоряжении ресурсов. При решении этого вопроса будем иметь в виду, что ресурсы, которые предприятие не может полностью использовать, имеют для него очень низкую ценность, в том смысле, что предприятие не будет нести даже небольшие расходы на увеличение запасов этих ресурсов. Заметим при этом, что рыночная цена этих ресурсов может быть большой (скажем дорогостоящее неиспользуемое или используемое не на полную мощность оборудование).

Наибольшую ценность будут иметь те ресурсы, которые в наибольшей степени ограничивают выпуск продукции, а следовательно и доход предприятия. Итак, можно считать, что каждый вид ресурса обладает некоторой, говорят, «теневой ценой», определяющей ценность данного ресурса для предприятия с точки зрения дохода от реализации выпускаемой продукции и зависящей от наличного запаса этого ресурса и потребности в нем для выпуска продукции.

Ясно, что варьируя план выпуска продукции, мы тем самым варьируем теневые цены используемых ресурсов, а потому ставится задача нахождения оптимальных теневых цен, которые соответствуют максимальному доходу предприятия, то есть оптимальному распределению ре-

сурсов. Оптимальные теневые цены называют объективно обусловленными или оптимальными оценками. Построим математическую модель для определения оптимальных теневых цен.

Обозначим через y_i теневую цену ресурса S_i . Величины y_i должны быть такими, чтобы теневая цена ресурсов, используемых в любом технологическом процессе, на производство единицы продукции не была меньше получаемого дохода от единицы продукции.

$$\sum_{i=1}^{m} a_{ij} y_j \geq c_j, \; j = \overline{1,l}. \qquad (1.39)$$

Оптимальными теневыми ценами естественно считать такие, которые минимизируют общую теневую стоимость ресурсов, то есть величину

$$g^* = \sum_{i=1}^{m} b_i y_i \rightarrow \min. \qquad (1.40)$$

Задача нахождения минимума целевой функции (1.40) при условиях (1.39) представляет собой задачу линейного программирования и называется двойственной по отношению к задаче (1.37)–(1.38).

1.4.2 Общая формулировка прямой и двойственной задачи

Дадим достаточно общее определение двойственной задачи по отношению к прямой, заданной в виде:

$$f = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \rightarrow \max \qquad (1.41)$$

при условиях

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n < R_1 > b_1 \\ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n < R_2 > b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n < R_m > b_m \\ x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0, \end{cases} \qquad (1.42)$$

где: R1,R2, …, R_n – операция отношения вида либо =, либо $\leq$ одновременно.

Очевидно, что любую задачу линейного программирования можно представить в одной из двух указанных форм. Тогда двойственной по отношению к задаче (1.41)–(1.42) будем называть задачу:

$$f^{*}=b_1y_1+b_2y_2+\ldots+b_my_m \to \min \qquad (1.43)$$

при условиях

$$\begin{cases} a_{11}y_1 + a_{21}y_2 +\ldots+ a_{m1}y_m \geq c_1 \\ a_{12}y_1 + a_{22}y_2 +\ldots+ a_{m2}y_m \geq c_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{1n}y_1 + a_{2n}y_2 +\ldots+ a_{mn}y_m \geq c_n \end{cases} \qquad (1.44)$$

и, если в прямой задаче $R_i \equiv (\leq)$, $i=\overline{1,n}$, то

$$y_1 \geq 0,\ y_2 \geq 0,\ldots,\ y_n \geq 0\ . \qquad (1.45)$$

Сформулируем правила построения двойственной задачи:

а) целевая функция исходной задачи (1.41)–(1.42) задается на максимум, а целевая функция двойственной задачи (1.43)–(1.45) – на минимум;

б) матрица системы сильных ограничений двойственной задачи получится из матрицы системы ограничений прямой задачи транспонированием;

в) число переменных двойственной задачи равно числу сильных ограничений прямой, а число ограничений в двойственной задаче равно числу переменных прямой;

г) коэффициентами при неизвестных в целевой функции двойственной задачи являются свободные члены системы сильных ограничений прямой задачи, а правыми частями системы сильных ограничений двойственной задачи – коэффициенты при неизвестных в целевой функции исходной задачи. Отметим, что если все $R_i \equiv (\leq)$ и, следовательно, в двойственной задаче выполнены условия (1.45) – переменные неотрицательны, то пара задач (1.41)–(1.42) и (1.43)–(1.45) называется симметричной.

1.4.3 Свойства двойственной задачи

Приведем теоремы, устанавливающие зависимости между решениями прямой и двойственной задач.

ТЕОРЕМА 1.2. Значение целевой функции задачи (1.41)–(1.42) при любом допустимом плане $\boldsymbol{x}$ не превышает значения целевой функции задачи (1.43)–(1.45) при любом ее допустимом плане $\mathbf{y}$: $f(\boldsymbol{x}) \le f^{*}(\mathbf{y})$.

Если для каких-нибудь $\boldsymbol{x}$ и $\mathbf{y}$ значения целевых функций совпадают, т.е. $f(\boldsymbol{x}) = f^{*}(\mathbf{y})$, то . $\boldsymbol{x}$- оптимальное решение (1.41)–(1.42), $\mathbf{y}$– оптимальное решение (1.43)–(1.45).

Доказательство. Пусть $\boldsymbol{x}=(x_1,x_2,\ldots,x_n)$ – допустимое решение задачи (1.41)–(1.42), $\mathbf{y}=(y_1,y_2,\ldots,y_m)$ – допустимое решение задачи (1.43)–(1.45). Подставим решение $\boldsymbol{x}$ в ограничения (1.42), умножим первое равенство на y_1, второе на $y_2,\ldots$, m–е на y_m и сложим полученные равенства:

$$(a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n)y_1+(a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n)y_2+\ldots$$
$$+(a_{m1}x_1+a_{m2}x_2+\ldots+a_{mm}x_n)y_m=b_1y_1+b_2y_2+\ldots+b_my_m.$$

Перегруппируем слагаемые в левой части:

$$(a_{11}y_1+a_{21}y_2+\ldots+a_{m1}y_m)x_1+(a_{12}y_1+a_{22}y_2+\ldots+a_{m2}y_m)x_2+\ldots$$
$$+(a_{1n}y_1+a_{2n}y_2+\ldots+a_{mn}y_m)x_n=b_1y_1+b_2y_2+\ldots+b_my_m. \qquad (1.46)$$

Так как $\mathbf{y}$ – допустимое решение задачи (1.43)–(1.45), то для любого j (j=1,2,…,n) , имеем:

$$a_{1j}y_1+a_{2j}y_2+\ldots+a_{mj}y_n \ge c_j.$$

Заменяя в (1.46) суммы, заключенные в скобки, соответствующими c_j и учитывая, что $x_j \ge 0$, получаем:

$$c_1x_1+c_2x_2+\ldots+c_nx_n \le b_1y_1+b_2y_2+\ldots+b_my_m,$$

или, что тоже самое $f(\boldsymbol{x}) \le f^{*}(\mathbf{y})$. Первая часть теоремы доказана.

Пусть для каких-нибудь $\boldsymbol{x},\mathbf{y}$ имеем: $(\boldsymbol{c},\boldsymbol{x}') > (\boldsymbol{b}^{\mathrm{T}},\mathbf{y})$. Допустим от противного, что $\boldsymbol{x}$ не является оптимальным решением задачи (1.41)–(1.42). Это значит, что есть такое другое допустимое решение $\boldsymbol{x}'$ задачи (1.41)–(1.42), что

$$(\boldsymbol{c},\boldsymbol{x}') > (\boldsymbol{c},\boldsymbol{x}) = (\boldsymbol{b}^{\mathrm{T}},\mathbf{y}),$$

а это противоречит первой части данной теоремы. Следовательно, при нашем условии $\boldsymbol{x}$– оптимальное решение задачи (1.41)–(1.42). Аналогично показывается, что **y** - оптимальное решение задачи (1.43)–(1.45).Теорема доказана.

ТЕОРЕМА 1.3. Если одна из пар двойственных задач (1.41)–(1.42) или (1.43)–(1.45) имеет оптимальное решение, то и другая также имеет оптимальное решение, причем их оптимумы совпадают ($\max f = \min f^*$).

Если целевая функция одной из пары двойственных задач не ограничена (для прямой – сверху, а для двойственной – снизу), то допустимая область другой задачи пуста.

Доказательство. Ввиду взаимной двойственности задач (1.41)–(1.42) и (1.43)–(1.45) достаточно провести доказательство, принимая за исходную одну из задач. Пусть, например, задача (1.41)–(1.42) имеет оптимальные решения, и $\boldsymbol{x}^{(0)}=(x_1^{(0)},x_2^{(0)},...,x_n^{(0)})$ – одно из них, найденное симплекс-методом, и $\boldsymbol{A}_{i_1},\boldsymbol{A}_{i_2},...,\boldsymbol{A}_{i_m}$ – его базис. Тогда на последней итерации имеем $\alpha_j=z_j-c_j\geq 0$ (j=1,2,…,n) так что $z_j \geq c_j$, j=1,2,…,n, или, в векторной форме

$$(z_1,z_2,...,z_n)\geq(c_1,c_2,...,c_n). \qquad (1.47)$$

При рассмотрении симплекс-метода мы выяснили, что вектор $(z_1,z_2,...,z_n)$ может быть найден по формуле

$$(z_1,z_2,...,z_n)=(c_{i_1},c_{i_2},...,c_{i_m})\,\boldsymbol{B}^{-1}\boldsymbol{A}=\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{A},$$

где $\boldsymbol{B}^{-1}$– матрица, обратная к базисной матрице $\boldsymbol{B}=(\boldsymbol{A}_{i_1},\boldsymbol{A}_{i_2},...,\boldsymbol{A}_{i_m})$, $\boldsymbol{A}$ –матрица системы ограничений-уравнений задачи (1.42): $\boldsymbol{A}=(\boldsymbol{A}_1,\boldsymbol{A}_2,...,\boldsymbol{A}_n)$.Следовательно, из (1.47) имеем:

$$(c_{i_1},c_{i_2},...,c_{i_m})\,\boldsymbol{B}^{-1}\boldsymbol{A}\ \geq(c_1,c_2,...,c_n)=\boldsymbol{c} \qquad (1.48)$$

Введем обозначение:

$$(c_{i_1},c_{i_2},...,c_{i_m})\boldsymbol{B}^{-1}=\mathbf{y}=(y_1,y_2,...,y_m).$$

Тогда неравенство (1.48) принимает вид

$$\mathbf{y}\boldsymbol{A}\geq\boldsymbol{c}.$$

Оно означает, что вектор $\mathbf{y}=\boldsymbol{c}_B\boldsymbol{B}^{-1}$ является допустимым решением задачи (1.43)-(1.45).

Докажем, что **y**– оптимальное решение задачи (1.43) – (1.45). Имеем:

$$(\mathbf{y},\boldsymbol{b})=((c_{i_1},c_{i_2},...,c_{i_m}),\ \boldsymbol{B}^{-1}\boldsymbol{b})=((c_{i_1},c_{i_2},...,c_{i_m}),(x_1^{(0)},x_2^{(0)},...,x_m^{(0)}))=$$
$$=c_{i_1}x_{i_1}^{(0)}+c_{i_2}x_{i_2}^{(0)}+...+c_{i_m}x_{i_m}^{(0)})=(\boldsymbol{c},\boldsymbol{x}^{(0)}).$$

Таким образом, значение целевой функции задачи (1.43) –(1.45) при её допустимом решении **y** совпадает с значением целевой функции задачи (1.41)–(1.42) при её допустимом решении $\boldsymbol{x}^{(0)}$. Согласно теореме 1.2 это означает, что **y** оптимальное решение задачи (1.43)–(1.45). Первая часть теоремы доказана.

Докажем вторую её часть. Пусть известно, что целевая функция, например, задачи (1.41)–(1.42) не ограничена сверху на допустимом множестве. Допустим, от противного, что задача (1.43)–(1.45) имеет при этом допустимое решение, и пусть **y** – одно из них. Ввиду неограниченности целевой функции задачи (1.41)–(1.42) найдётся такое допустимое её решение $\boldsymbol{x}$, для которого значение целевой функции будет больше числа $(\boldsymbol{b},\mathbf{y})$:

$$(\boldsymbol{c},\boldsymbol{x})>(\boldsymbol{b},\mathbf{y}),$$

что противоречит теореме 1.2. Теорема доказана.

При доказательстве первой части теоремы мы установили следующее: если $\boldsymbol{B}$ – базисная матрица оптимального опорного решения канонической задачи (1.41)–(1.42), то

$$\mathbf{y}=\boldsymbol{c}_B\boldsymbol{B}^{-1}$$

– оптимальное решение двойственной к (1.41)–(1.42) задачи (1.43)-(1.45). Этим фактом мы будем пользоваться в дальнейшем.

Рассмотрим, в качестве примера, следующую задачу и двойственную к ней.

$$f = 5x_1 + 10x_2 \to \max \qquad f^* = 96y_1 + 144y_2 + 48y_3 \to \min$$

$$\begin{cases} 4x_1 + 3x_2 \le 96 \\ 5x_1 + 8x_2 \le 144 \\ x_1 + 4x_2 \le 48 \\ x_1 \ge 0, \quad x_2 \ge 0. \end{cases} \qquad \begin{cases} 4y_1 + 5y_2 + y_3 \ge 5 \\ 3y_1 + 8y_2 + 4y_3 \ge 10 \\ y_1 \ge 0,\ y_2 \ge 0,\ y_3 \ge 0. \end{cases}$$

Оптимальное решение прямой задачи, приведенное в последней симплекс таблице: $\boldsymbol{x}^{\mathrm{T}}=(16,8,8,0,0)$ получено в базисе $\boldsymbol{A}_3$, $\boldsymbol{A}_1$, $\boldsymbol{A}_2$. Найдем матрицы $\boldsymbol{B}$, $\boldsymbol{B}^{-1}$

$$\boldsymbol{B} = \begin{pmatrix} 1 & 4 & 3 \\ 0 & 5 & 8 \\ 0 & 1 & 4 \end{pmatrix}; \qquad \boldsymbol{B}^{-1} = (1/12)\begin{pmatrix} 12 & -13 & 17 \\ 0 & 4 & -8 \\ 0 & -1 & 5 \end{pmatrix};$$

$$\mathbf{y}^* = \boldsymbol{c}_{\mathrm{B}}\, \boldsymbol{B}^{-1} = (0,\ 5/6,\ 5/6);$$
$$\max f = f(\boldsymbol{x}^*) = 160 = \min f^*(\mathbf{y}^*).$$

В том случае, когда среди векторов $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, …, $\boldsymbol{A}_{\mathrm{n}}$ прямой задачи(1.41)–(1.42) имеется m – единичных $(\boldsymbol{A}_{\mathrm{j1}}, \boldsymbol{A}_{\mathrm{j2}}, \ldots, \boldsymbol{A}_{\mathrm{jm}})$, решение двойственной задачи может быть найдено проще, а именно, компоненты вектора

$$y_k = \alpha_{jk} + c_{jk},\ k = \overline{1,m};$$
$$\mathbf{y}^* = (y_1, \ldots, y_m)^{\mathrm{T}},$$

где α_{jk} – элементы строки оценок в последней симплекс-таблице решения прямой задачи, а c_{jk} – соответствующие коэффициенты целевой функции прямой задачи.

Рассмотрев предыдущий пример, можем убедиться, что поскольку в исходной задаче векторы $\boldsymbol{A}_3$, $\boldsymbol{A}_4$, $\boldsymbol{A}_5$ - единичные, то:

$$\begin{cases} y_1 = \alpha_3 + c_3 = 0 + 0 = 0 \\ y_2 = \alpha_4 + c_4 = 5/6 + 0 = 5/6 \\ y_3 = \alpha_5 + c_5 = 5/6 + 0 = 5/6 \end{cases}$$
$$\mathbf{y}^* = (0;\ \ 5/6;\ \ 5/6).$$

1.4.4 Анализ чувствительности

Нас будет интересовать вопрос: какое воздействие окажут изменения, вносимые в те или иные параметры задачи линейного программирования, на ее решение, на решение двойственной задачи, на оптимальные значения целевой функции ? А именно, изучим:

а) воздействие дополнительного количества дефицитного ресурса;

б) воздействие дополнительного количества недефицитного ресурса;

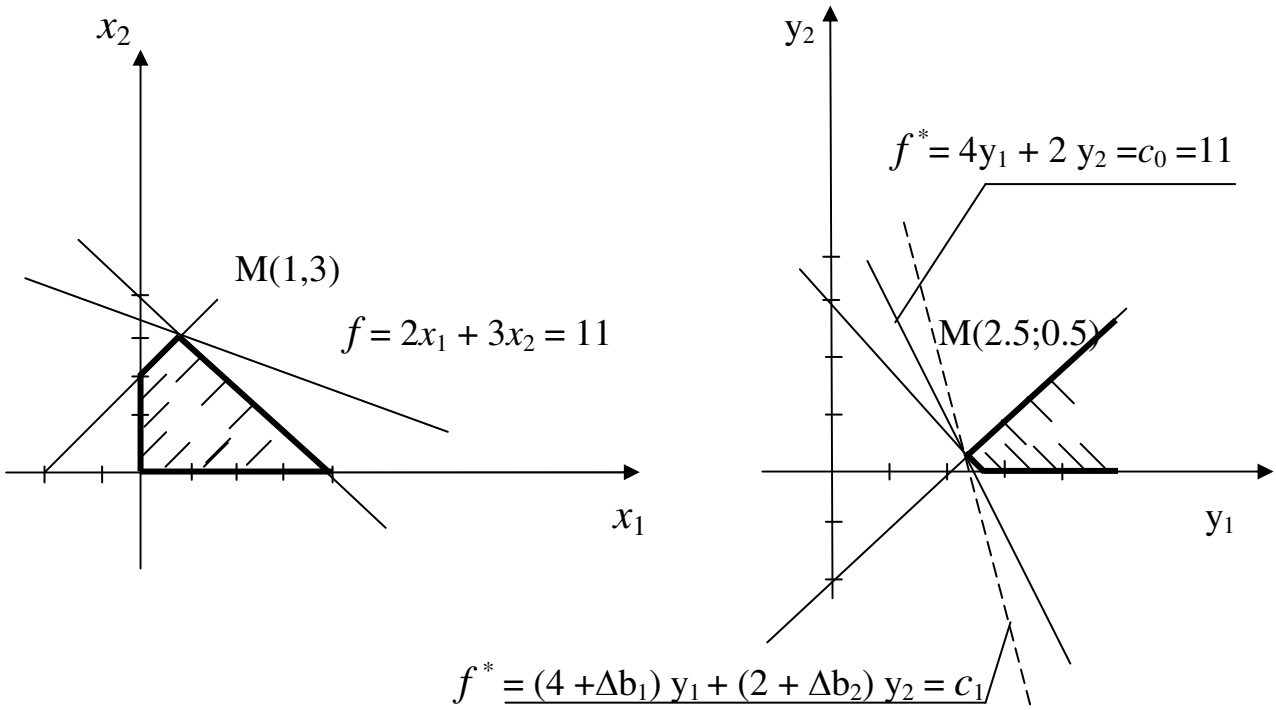
в) воздействие изменений в коэффициентах целевой функции.

Для наглядности исследований рассмотрим прямую и двойственную задачи линейного программирования и их геометрическое решение (рисунок 1.5).

Пример 1.5.

$$f = 2x_1 + 3x_2 \to \max$$
$$\begin{cases} x_1 + x_2 \le 4 \\ -x_1 + x_2 \le 2 \end{cases}$$
$$x_1 \ge 0,\ x_2 \ge 0$$

$$f^* = 4y_1 + 2y_2 \to \min$$
$$\begin{cases} y_1 - y_2 \ge 2 \\ y_1 + y_2 \ge 3 \end{cases}$$
$$y_1 \ge 0,\ y_1 \ge 0$$



$$f_{\max} = f\Big|_{\substack{x_1=1\\ x_2=3}} = 11 \qquad f^*_{\min} = f^*\Big|_{\substack{y_1=2.5\\ y_2=0.5}} = 11$$

Рисунок 1.5 - Решение задачи линейного программирования геометрическим способом.

Приведем заключительную симплекс-таблицу решения прямой задачи.

Таблица 1.6 – Заключительная симплекс-таблица

Базис	c_B	b опорн.		c_1	c_2	c_3	c_4
				2	3	0	0
				A_1	A_2	A_3	A_4
A_1	2	1		1	0	1/2	–1/2
A_2	3	3		0	1	1/2	1/2
		$f = 11$	α_j	0	0	2,5	0,5
						y_1	y_2

Очевидно, что значение целевой функции задачи линейного программирования $f = f(b_1, b_2, \ldots, b_m)$ зависит, вообще говоря, от запасов ресурсов. Изменение ресурсов b_i на малую величину Δb_i, i = 1,2,...,m ведет к изменению значения max f =min f^*, но изменение min f^* происходит,

при малых Δb_i, не за счет изменения точки, в которой достигается $\min f^*$ (допустимая область двойственной задачи вообще не изменится при изменении b_i), а за счет изменения наклона поверхности (линии) уровня целевой функции f^*, коэффициенты которой зависят от $b_i+\Delta b_i$.

Следует отметить, что при больших изменениях b_i, $i=\overline{1,m}$ или даже некоторых из них, оптимальное решение двойственной задачи может перейти в другую угловую точку. Нас будет интересовать в каких пределах изменение запасов b_i на величину Δb_i не влечет изменения двойственных оценок (теневых цен). Множество таких значений Δb_i будем называть областью устойчивости решения двойственной задачи.

Рассмотрим влияние на устойчивость двойственных оценок приращений Δb_i дефицитных и недефицитных ресурсов. Очевидно, что увеличение недефицитных ресурсов не приведет к увеличению прибыли, а появление дополнительного количества дефицитных ресурсов улучшает, вообще говоря, оптимальное значение целевой функции. Однако необходимо принять во внимание, что изменение оптимального решения за счет увеличения количества дефицитных ресурсов приведет к улучшению значения целевой функции (прямой задачи) только в том случае, если сумма дополнительных издержек по обеспечению дополнительным количеством ресурса не превышает сумму прибыли, полученной в результате его использования.

В рассмотренном примере увеличение Δb_2 на величину большую 2, то есть $b_2+\Delta b_2>4$, при фиксированном $b_1=4$, перемещает оптимальное решение двойственной задачи в точку $y_1=3$, $y_2=0$ и $\min f^* = \max f = 12$. Таким образом, увеличение на 1 единицу произошло за счет увеличения ресурса b_2 на величину ≥ 2. Причем, начиная со значения $b_2=4$, этот ресурс перестает быть дефицитным. При $0<\Delta b_2<2$ и $\Delta b_1>0$ точка минимума f^* не меняется, а значение целевой функции f^* увеличивается, а следовательно, увеличивается и значение целевой функции прямой задачи. Это и есть область устойчивости двойственных оценок дефицитных ресурсов. Здесь же отметим, что

если b_1=4, b_2>4, то уменьшая недефицитный ресурс b_2 на величину Δb_2<0, мы при $b_2+\Delta b_2$<4 добьемся того, что оптимальное решение двойственной задачи будет достигаться в точке (2,5; 0,5) и ресурс y_2 станет дефицитным, но значение $\min f^* = \max f$ уменьшится.

В общем случае рассмотрим иной подход к изучению влияния Δb_i на область устойчивости двойственных оценок. Пусть $\boldsymbol{B}$– матрица векторов исходной задачи, составляющих базис, в котором решение прямой задачи оптимально. Тогда последняя симплекс-таблица 1.7 формально может быть получена так:

Таблица 1.7 – Итоговая симплекс-таблица

Базис	$\boldsymbol{c}_B$	Опорное решение	c_1	c_2	…	c_n
			$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	…	$\boldsymbol{A}_n$
$\boldsymbol{A}_{i_1}$ … $\boldsymbol{A}_{i_m}$	c_{i_1} ... c_{i_m}	$\boldsymbol{B}^{-1}\boldsymbol{b}$	$\boldsymbol{B}^{-1}\boldsymbol{A}_1$	$\boldsymbol{B}^{-1}\boldsymbol{A}_2$	…	$\boldsymbol{B}^{-1}\boldsymbol{A}_n$
Оценка			α_1	α_2	…	α_n

Очевидно, что изменения запасов ресурсов на величины Δb_i не окажет влияния ни на разложения векторов $\boldsymbol{A}_1$,…, $\boldsymbol{A}_n$ по базису, ни на значения $\boldsymbol{A}_j$, а стало быть и на значения двойственных оценок, если только $\boldsymbol{B}^{-1}(\boldsymbol{b}+\Delta\boldsymbol{b})$, будет оставаться допустимым опорным решением задачи линейного программирования, т.е. его компоненты будут оставаться неотрицательными. Условия неотрицательности компонент вектора $\boldsymbol{B}^{-1}(\boldsymbol{b}+\Delta\boldsymbol{b})$ позволит определить область устойчивости двойственных оценок и выяснить порознь влияние дополнительного количества дефицитных и недефицитных ресурсов. В рассматриваемом примере

$$\boldsymbol{B}=\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix},\quad \boldsymbol{b}=\begin{pmatrix}4\\ 2\end{pmatrix},\quad \boldsymbol{B}^{-1}=\begin{pmatrix}1/2 & -1/2\\ 1/2 & 1/2\end{pmatrix}.$$

Оптимальное опорное решение (значения базисных переменных):

$$\boldsymbol{B}^{-1}\boldsymbol{b} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}.$$

Пусть мы внесли изменения в вектор правых частей (увеличили запасы ресурсов)

$$\boldsymbol{b} + \Delta\boldsymbol{b} = \begin{pmatrix} 4 + \Delta b_1 \\ 2 + \Delta b_2 \end{pmatrix},$$

тогда оптимальное опорное решение новой задачи (значения базисных переменных):

$$\boldsymbol{B}^{-1}(\boldsymbol{b} + \Delta\boldsymbol{b}) = \begin{pmatrix} 2 + \frac{1}{2}\Delta b_1 - 1 - \frac{1}{2}\Delta b_2 \\ 2 + \frac{1}{2}\Delta b_1 + 1 + \frac{1}{2}\Delta b_2 \end{pmatrix} = \begin{pmatrix} 1 + \frac{1}{2}\Delta b_1 - \frac{1}{2}\Delta b_2 \\ 3 + \frac{1}{2}\Delta b_1 + \frac{1}{2}\Delta b_2 \end{pmatrix}$$

при условиях:

$$\begin{cases} 1 + (1/2)\Delta b_1 - (1/2)\Delta b_2 \geq 0 \\ 3 + (1/2)\Delta b_1 + (1/2)\Delta b_2 \geq 0 \\ \Delta b_1 \geq 0,\ \Delta b_2 \geq 0. \end{cases}$$

Область устойчивости двойственных оценок в этом случае нетрудно изобразить геометрически (рисунок 1.6).

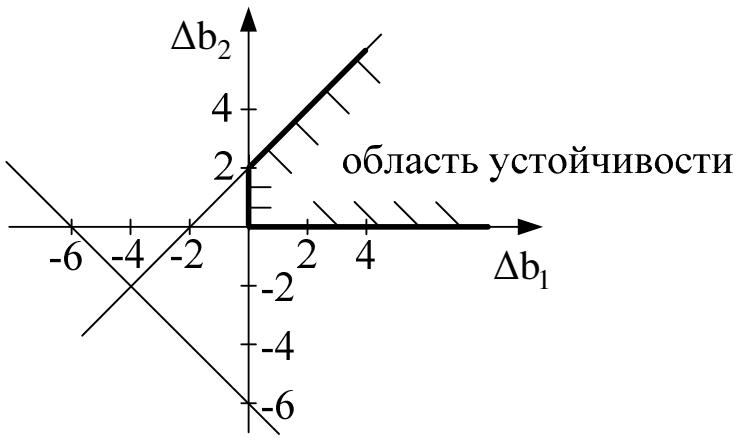


Рисунок 1.6 – Область устойчивости

Итак, область устойчивости: Δb_1 – любое неотрицательное и Δb_2 может принимать значения $0 \le \Delta b_2 \le 2+\Delta b_1$. В частности, при $\Delta b_1 = 0$, $0 \le \Delta b_2 \le 2$.

В задачах большей размерности мы сможем, вообще говоря, лишь проверять попадают ли компоненты вектора $\boldsymbol{B}^{-1}(\boldsymbol{b}+\Delta b)$ в область устойчивости.

Обратимся к вопросу влияния вариации коэффициентов целевой функции $(c_1,\ldots,c_n)$ прямой задачи на оптимальное решение. В частности нас интересует вопрос устойчивости решения прямой задачи относительно вариации коэффициентов целевой функции. Легко видеть, что вариация $c_1,\ldots,c_n$ не оказывает влияния на допустимую область задачи линейного программирования и нас интересует область изменения Δc_1, $\Delta c_2,\ldots$, Δc_n , в которой точка достижения оптимального решения не изменится. Очевидно, что если рассматривать в качестве прямой задачи двойственную, то решение проблемы будет аналогично рассмотренному выше (роль $b_1,\ldots,b_m$ будут играть $c_1,\ldots, c_n$). Например, в рассмотренном примере базисом оптимального решения двойственной задачи будут естественно векторы $\boldsymbol{A}_1=(1,1)^T$ и $\boldsymbol{A}_2=(-1,1)^T$. Оптимальное опорное решение (значение базисных переменных)

$$\boldsymbol{B}^{-1}\boldsymbol{c} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 0,5 & 0,5 \\ -0,5 & 0,5 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2,5 \\ 0,5 \end{pmatrix}.$$

Для изучения устойчивости внесем изменения в вектор цен

$$\boldsymbol{c} + \Delta\boldsymbol{c} = (2 + 3\Delta\boldsymbol{c}_1 + \Delta\boldsymbol{c}_2).$$

Найдем оптимальное опорное решение новой двойственной задачи (значение базисных переменных).

$$\boldsymbol{B}^{-1}(\boldsymbol{c}+\Delta\boldsymbol{c}) = \begin{pmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 2+\Delta c_1 \\ 3+\Delta c_2 \end{pmatrix} = \begin{pmatrix} 1+\frac{1}{2}\Delta c_1 + \frac{3}{2} + \frac{1}{2}\Delta c_2 \\ -1-\frac{1}{2}\Delta c_1 + \frac{3}{2} + \frac{1}{2}\Delta c_2 \end{pmatrix}.$$

Потребуем:

$$5/2 + (1/2)\Delta c_1 + (1/2)\Delta c_2 \ge 0$$
$$1/2 - (1/2)\Delta c_1 + (1/2)\Delta c_2 \ge 0$$
$$\Delta c_1 \ge 0,\ \Delta c_2 \ge 0,$$

что определяет область устойчивости решения прямой задачи относительно вариации коэффициентов (см. рисунок 1.7).

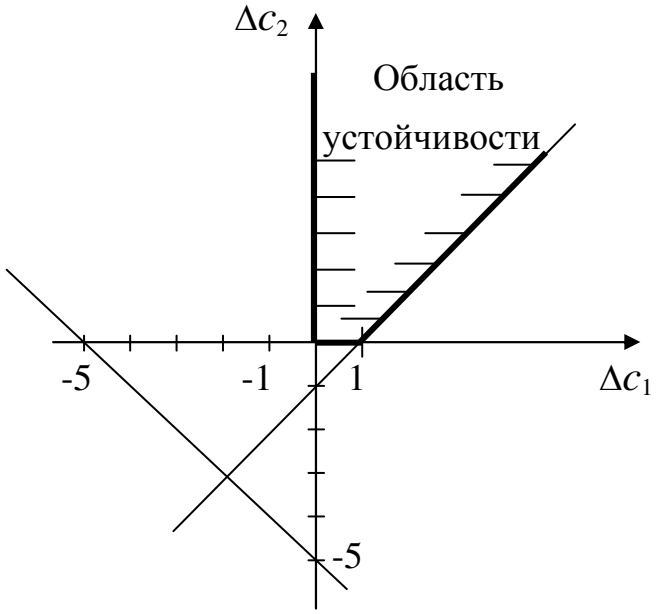


Рисунок 1.7 – Область устойчивости

1.4.5 Экономическая интерпретация двойственных задач

Экономическую интерпретацию двойственных задач и двойственных оценок рассмотрим на примере.

Для производства продукции двух видов A и B используются ресурсы трех видов I,II,III. Запасы ресурсов – b_i, затраты ресурсов a_{ij} на производство единицы продукции каждого вида и цена c_j единицы продукции приведены в таблице 1.8.

Таблица 1.8 – Таблица выпускаемой продукции

Ресурсы	A	B	Запасы ресурсов
I	1	1	10
II	2	4	30
III	1	0	8
c_i	2	1	

Требуется так спланировать производство, чтобы прибыль от реализации продукции была максимальной. Оценить теневые цены ресурсов и произвести анализ решения двойственной задачи. Обозначим $x_i, i=\overline{1,3}$ план выпуска продукции A, B, C, y_i – теневые цены ресурсов I,II,III типа. Тогда прямая и двойственная задачи имеют вид:

$$f = 2x_1 + x_2 \to \max \quad \begin{cases} x_1 + x_2 \le 10 \\ 2x_1 + 4x_2 \le 30 \\ x_1 \le 8 \\ x_1 \ge 0 \\ x_3 \ge 0 \end{cases}$$

$$f^* = 10y_1 + 30y_2 + 8y_3 \to \min \quad \begin{cases} y_1 + 2y_2 + y_3 \ge 2 \\ y_1 + 4y_2 + 0y_3 \ge 1 \\ y_1 \ge 0 \\ y_2 \ge 0 \\ y_3 \ge 0. \end{cases}$$

Решение прямой и двойственной задач приведено в заключительной симплекс-таблице 1.9.

Таблица 1.9 – Симплекс-таблица

Базис	$\boldsymbol{c}_B$	$\boldsymbol{b}$	2	1	0	0	0
			$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	$\boldsymbol{A}_3$	$\boldsymbol{A}_4$	$\boldsymbol{A}_5$
$\boldsymbol{A}_1$	2	8	1	0	0	0	1
$\boldsymbol{A}_2$	1	2	0	1	1	0	–1
$\boldsymbol{A}_4$	0	4	0	0	–4	1	2
Оценки		f=18	0	0	1	0	1

Оптимальный план производства продукции вида A: x_1=8; вида B: x_2=2; f_{max}=18. Ресурсы типа I и III имеют теневые цены, большие нуля, поэтому дефицитны и в оптимальном плане используются полностью, что легко проверить по 1-му и 3-му ограничениям прямой задачи.
Ресурс II типа имеет теневую цену равную нулю, является недефицитным и в оптимальном плане недоиспользуется в объеме 6 ед., что легко вычисляется из 2-го ограничения прямой задачи. Увеличение запасов дефицитного ресурса I типа ведет к увеличению прибыли от дополнительно произведенной продукции на одну единицу за счет следующих изменений в плане выпуска продукции: выпуск продукции вида B увеличивается на одну едини-

цу, выпуск продукции вида А остается на прежнем уровне, а использование ресурса II типа увеличивается на 4 единицы (результаты получены из компонент столбца $\boldsymbol{A}_3$ и легко проверяются непосредственной проверкой). По аналогии при увеличении запасов дефицитного ресурса III типа на одну единицу прибыль от реализации дополнительно произведенной по новому оптимальному плану продукции также возрастает на одну единицу, при этом, как видно из последнего столбца симплекс – таблицы 1.9 необходимо увеличить выпуск продукции вида А на одну единицу и одновременно выпуск продукции вида В сократить на одну единицу. Использование ресурса II типа уменьшится еще на 2 единицы.

2. Специальные задачи линейного программирования

2.1 Транспортная задача

Общая постановка транспортной задачи состоит в определении оптимального плана перевозок однородного груза из N пунктов отправления $A_1,\ldots, A_N$ в M пунктов потребления $B_1, B_2,\ldots, B_M$. При этом в качестве критерия оптимальности обычно берется минимальная стоимость перевозок всего груза. Мощности поставщиков и запросы потребителей, а также затраты на перевозку единицы груза для каждой пары «поставщик-потребитель» будем сводить в таблицу поставок.

Рассмотрим, к примеру, следующую задачу.

Имеется N карьеров, где добывается песок и M потребителей песка (кирпичные заводы, растворобетонные узлы и т.д.). В i-ом карьере ежесуточно добывается a_i тонн песка, а j-му потребителю ежесуточно требуется b_j тонн песка. Стоимость перевозки одной тонны песка с i-го карьера на j-ый пункт потребления равна c_{ij}.

Требуется составить план перевозок песка таким образом, чтобы все пункты потребления были снабжены требуемым количеством песка, на карьерах не должно создаваться запасов добытого песка, а стоимость перевозок была бы минимальной.

Предположим x_{ij} - это количество тонн сырья, которое было перевезено с i-го карьера на j-ый пункт потребления. Суммарная стоимость перевозок, будет равна

$$f = \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}x_{ij}, \qquad (2.1)$$

при этом на j-ый пункт потребления должно быть доставлено b_j количество песка, т.е.

$$\sum_{i=1}^{N} x_{ij} = b_j,$$

а с i-го карьера вывезено a_i, тонн песка, т.е.,

$$\sum_{j=1}^{M} x_{ij} = a_i .$$

Потребуем также, чтобы все $x_{ij} \geq 0$. Итак, получили следующую задачу:

$$f = \sum_{i=1}^{N} \sum_{j=1}^{M} c_{ij} x_{ij} \to \min .$$

Найти минимальное значение f при выполнении условий:

$$\sum_{i=1}^{N} x_{ij} = b_j \ (j = \overline{1,M}\ ),$$

$$\sum_{j=1}^{M} x_{ij} = a_i \ (i = \overline{1,N}\ ), \qquad (2.2)$$

$$x_{ij} \geq 0, \ (\ i = \overline{1,N};\ j = \overline{1,M}\ ). \qquad (2.3)$$

Как мы видим, это задача линейного программирования, и поиск оптимального плана транспортной задачи можно вести с помощью симплекс-метода. Однако, в силу некоторых свойств этой задачи (каждая неизвестная входит лишь в два уравнения-ограничения и коэффициенты при неизвестных в ограничениях равны единице), она может проще решаться специальными методами.

ПРИМЕЧАНИЕ. Термин "транспортная задача" (в дальнейшем ТЗ) возник в 30-х годах. Это была одна из самых первых задач, которую стали решать с помощью методов линейного программирования. Множество проблем, совершенно не связанных с перевозкой грузов могут иметь своей математической моделью "транспортную задачу".

Перейдем к рассмотрению способов решения транспортной задачи.

ОПРЕДЕЛЕНИЕ 2.1 Транспортная задача, в которой сумма запасов равна сумме потребностей, называется закрытой. В противном случае задача – открытая.

В случае если транспортная задача является открытой, невозможно удовлетворить всех потребителей (если сумма потребностей больше суммы запасов) или вывезти все грузы от поставщиков (если сумма запасов больше, чем сумма потребностей).

В этом случае поступают следующим образом:

а) если сумма запасов больше суммы потребностей

$$(\sum_{i=1}^{N} a_i > \sum_{j=1}^{M} b_j),$$

то введем в таблицу еще одного потребителя, потребность которого определим, как

$$\sum_{i=1}^{N} a_i - \sum_{j=1}^{M} b_j .$$

Так как грузы к новому потребителю (фиктивному) отправляться не будут, то тарифы на перевозку грузов фиктивному потребителю положим равными нулю;

б) если сумма запасов меньше суммы потребностей

$$(\sum_{i=1}^{N} a_i < \sum_{j=1}^{M} b_j),$$

то вводим в таблицу еще одного поставщика (фиктивного), запасы груза у которого определим, как

$$\sum_{j=1}^{M} b_j - \sum_{i=1}^{N} a_i ,$$

тарифы на перевозку грузов от фиктивного поставщика потребителям положим равными нулю из тех же соображений, что и в первом случае (цены в новом столбце проставим равными нулю).

Из чего следует, что в дальнейшем можем рассматривать только закрытые задачи. Приведем без доказательства теорему.

ТЕОРЕМА 2.1. Необходимым и достаточным условием разрешимости транспортной задачи является ее закрытость.

Так как транспортная задача является задачей линейного программирования, то и схема нахождения оптимального решения остается той же: находится первоначальный опорный план, проверяется на оптимальность и, если он не оптимален, то переходим к другому опорному плану, улучшающему целевую функцию в смысле оптимума (например, уменьшающему значение целевой функции).

Вернемся к общей постановке транспортной задачи (2.1)–(2.3). Мы имеем M x N неизвестных x_{ij} при $i=\overline{1,N}$ и $j=\overline{1,M}$ и $M + N$ уравнений. Система ограничений (2.2) линейно зависима. Сложив первые M уравнений системы (2.2), получим:

$$\sum_{j=1}^{M}\sum_{i=1}^{N} x_{ij} = \sum_{j=1}^{M} b_j .$$

Сумма последних N уравнений дает

$$\sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} = \sum_{i=1}^{N} a_i .$$

Но для закрытой транспортной задачи

$$\sum_{i=1}^{N} a_i = \sum_{j=1}^{M} b_j , \qquad \text{откуда следует}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} = \sum_{j=1}^{M}\sum_{i=1}^{N} x_{ij} .$$

Следовательно, она может быть разрешена относительно не более чем $M+N$-1 неизвестной (можно доказать, что ранг системы ограничений (2.2) в точности равен $M+N-1$), то есть опорный план может иметь не более $M+N-1$ отличных от нуля неизвестных.

ОПРЕДЕЛЕНИЕ 2.2. Опорный план, содержащий $M+N-1$ отличных от нуля значений неизвестных, называется невырожденным, а в противном случае – вырожденным.

2.2 Поиск начального опорного плана

Для нахождения опорного плана существуют несколько методов (метод северо-западного угла, метод минимального элемента, метод аппроксимации Фогеля). Рассмотрим методы северо - западного угла и минимального элемента.

2.2.1 Метод северо-западного угла

В верхнюю левую клетку (северо-западный угол) таблицы поставок записываем наименьшее из чисел b_1 и a_1, пересчитываем запасы и потребности и столбец, с исчерпанным запасом, или строку с удовлетворенной потребностью исключаем из дальнейшего расчета. В оставшейся части таблицы снова находим северо-западный угол, заполняем эту клетку, вычеркиваем строку или столбец и опять обращаемся к северо-западному углу и так далее. Важнейшим условием построения опорного плана является назначение в выбранной клетке наибольшего возможного плана перевозки.

Пример 2.1. Три песчано-гравийных карьера добывают в сутки 140, 180 и 160 условных единиц гравия. Для строительства пяти дорог необходимо гравия в количестве 60, 70, 120, 130 и 100 условных единиц соответственно. Стоимость перевозок (тарифов) из одного карьера на один объект (строящуюся дорогу) приведена в таблице 2.1 в условных денежных единицах (например, чтобы перевезти 1 условную единицу гравия с карьера №1 на дорогу №1 надо затратить две условные денежные единицы).

Составим таблицу 2.1.

Таблица 2.1

Карьер / Дорога	Карьер № 1	Карьер № 2	Карьер № 3	Потребности
№1	2 /60	8	9	60
№2	3	5	8	70
№3	4	1	4	120
№4	2	4	7	130
№5	4	1	2	100
Запасы	140	180	160	480/480

В северо-западную клетку поместим число 60 (дороге №1 требуется 60 условных единиц гравия). После такого распределения карьер №1 может поставить еще 80 условных единиц (140−60=80), а первую строку в дальнейшем не рассматриваем (дорогу №1 мы уже обеспечили).

Переходим к следующей "северо-западной" клетке – это клетка (дорога №2; карьер №1). В нее поместим 70 единиц (min (80;70)=70, где 80 – оставшийся запас на карьере №1, а 70 – потребность дороги №2). "Вычеркнув" вторую строку, перейдем к клетке (дорога №3; карьер №1). В эту клетку мы сможем поместить только 10 единиц, так как у карьера №1 осталось всего 10 единиц (140−60−70). После этого переходим к клетке (дорога №3; карьер №2), предварительно "вычеркнув" первый столбец и так далее.

В результате получаем таблицу 2.2.

Таблица 2.2

Карьер / Дорога	Карьер № 1	Карьер №2	Карьер №3	Потребности
№1	2 /60	8	9	60
№2	3/70	5	8	70
№3	4 /10	1 /110	4	120
№4	2	4 /70	7 /60	130
№5	4	1	2 /100	100
Запасы	140	180	160	480 /480

Найдем затраты на перевозки при составленном плане:

$f = 2\cdot60+8\cdot0+9\cdot0+3\cdot70+5\cdot0+8\cdot0+4\cdot10+1\cdot110+4\cdot0+$
$+2\cdot0+4\cdot70+ 7\cdot60+4\cdot0+1\cdot0+2\cdot100=1380$ условных денежных единиц.

Приведем свойство опорного плана, который получен с помощью метода северо – западного угла.

ТЕОРЕМА 2.2. Число положительных компонент в опорном плане (число заполненных клеток в таблице) меньше или равно $N+M-1$.

Действительно: пусть имеем M потребителей и N поставщиков. В процессе построения опорного плана на каждом шаге вносим в одну клетку (i, j) положительное число. При этом либо в одной строке, либо в одном столбце потребности или запасы после пересчета становятся равными нулю.

При заполнении последней клетки после пересчета и в столбце и в строке запасы и потребности становятся равными нулю. Итак, мы получили M нулей в пересчитанном столбце потребностей и N нулей в пересчитанной строке запасов. Так как ноль в строке или (возможно «и») столбце мы получаем в результате заполнения одной клетки, а при заполнении последней клетки мы получаем сразу два нуля, то число заполненных клеток равно $M+N-1$ или меньше.

Если в процессе построения плана встретится клетка (кроме последней), после заполнения которой запасы и потребности столбца и строки становятся равными нулю, то число неизвестных будет меньше $N+M-1$. И мы будем иметь вырожденный опорный план.

2.2.2 Метод минимального элемента

Алгоритм нахождения начального опорного плана.

1. В клетку с минимальной единичной стоимостью (минимальным тарифом) записывают наибольшее возможное количество груза для поставки.
2. Производится корректировка оставшихся запасов и потребностей.

3. Выбирается следующая клетка с наименьшим тарифом, в которую планируется наибольшее возможное количество груза для поставки и т.д. до тех пор, пока оставшиеся запасы и оставшиеся потребности не станут равными нулю.
4. Если наименьший тариф соответствует более чем одной клетке, выбор осуществляется случайным образом.

Для предыдущего примера начальный опорный план, построенный методом наименьшей стоимости, приводится в таблице 2.3.

Таблица 2.3

Карьер / Дорога	Карьер №1	Карьер №2	Карьер №3	Потребности
№1	2/60	8	9	60
№2	3	5	8 /70	70
№3	4	1/120	4	120
№4	2/80	4	7/50	130
№5	4	1/60	2/40	100
Запасы	140	180	160	480/480

2.2.3 Решение транспортной задачи методом потенциалов

Изложенный ниже метод потенциалов решения транспортной задачи реализуется для невырожденных опорных решений. Поэтому, если начальный опорный план окажется вырожденным (заполнено меньше, чем $M+N-1$ клеток), то для соответствующего потребителя требуется ввести дополнительно нулевую поставку (или сколь угодно малую) от следующего поставщика, увеличив тем самым число формально занятых клеток. Это может случиться в силу того, что при составлении опорного плана на каких-то этапах из рассмотрения выпадали одновременно строка и столбец (удовлетворены потребности и использованы запасы).

Исследуем опорный план задачи на оптимальность. Вначале припишем каждому потребителю величину

$v_j, j=\overline{1,M}$ -потенциал j-го потребителя, который при данном опорном плане характеризует затраты на размещение одной единицы поставляемой продукции указанному потребителю. Каждому поставщику припишем величину u_i , $i=\overline{1,N}$ - потенциал i-го поставщика, который при данном опорном плане характеризует затраты на поставку одной единицы продукции от указанного поставщика.

Очевидно, что величина $v_j - u_i$ – характеризует затраты на поставку от i-го поставщика j-му потребителю. Определим ее из условия $v_j - u_i = c_{ij}$ для занятых клеток. Тогда разность $(v_j - u_i) - c_{ij}$, найденная для свободных клеток, будет характеризовать изменение затрат на транспортировку одной единицы груза, при изменении маршрута транспортировки в соответствующую свободную клетку при найденном распределении потенциалов.

Величину $v_j - u_i - c_{ij}$ будем называть теневой ценой по перемещению единицы груза от i-го поставщика j-му потребителю. Если теневая цена для свободной клетки меньше нуля, то перемещение по маршруту i→j может лишь привести к увеличению затрат, что нецелесообразно. Если же теневая цена для свободной клетки больше нуля, то перемещение по маршруту приведет к уменьшению затрат. В итоге можно сформулировать условия оптимальности опорного плана.

ТЕОРЕМА 2.3. Если для некоторого опорного плана транспортной задачи будут выполняться условия $v_j - u_i - c_{ij} = 0$ для клеток с $x_{ij} > 0$ и $v_j - u_I - c_{ij} \le 0$ для клеток с $x_{ij} = 0$, то этот план является оптимальным.

Проверим оптимальность полученного плана в приведенной выше задаче. Сначала найдем потенциалы u_i и v_j. Установим соответствие:

K_1=> j=1; K_2=> j=2; K_3=> j=3; Дорога № 1=> i=1,…, дорога № 5=> i=5.

Тогда клетки, в которых $x_{ij} > 0$ будут следующие: (1,1)=60; (2,1)=70; (3,1)=10; (3,2)=110; (4,2)=70; (4,3)=60; (5,3)=100. Составим уравнения для нахождения u_i и v_j:

$$
\begin{aligned}
&v_1 - u_1 - 2 = 0 \quad \text{(2 - цена (тариф) для клетки (1,1))}\\
&v_1 - u_2 - 3 = 0\\
&v_1 - u_3 - 4 = 0\\
&v_2 - u_3 - 1 = 0\\
&v_2 - u_4 - 4 = 0\\
&v_3 - u_4 - 7 = 0\\
&v_3 - u_5 - 2 = 0.
\end{aligned} \qquad (2.4)
$$

Получили систему из семи уравнений с восемью неизвестными (в общем случае $M+N-1$ уравнение по числу заполненных клеток) и $M+N$ неизвестных (по числу потенциалов). Такая система уравнений имеет бесчисленное множество решений. Придавая одной из переменных какое-то значение (например, $u_1=0$) определяем остальные потенциалы из решения системы

$u_1=0$; $v_1=2$; $u_2=-1$; $u_3=-2$; $v_2=-1$; $u_4=-5$; $v_3=2$; $u_5=0$.

Теперь проверяем наш план на оптимальность, то есть проверяем условие $v_i - u_i - c_{ij} \le 0$ для незаполненных клеток:

$$
\begin{aligned}
&(1,2): -1 - 0 - 8 = -9 \le 0\\
&(1,3): 2 - 0 - 9 = -7 \le 0\\
&(2,2): -1 - (-1) - 5 = -5 \le 0\\
&(2,3): 2 - (-1) - 8 = -5 \le 0\\
&(3.3): 2 - (-2) - 4 = 0\\
&(4,1): 2 - (-5) - 2 = 5\\
&(5,1): 2 - 0 - 4 = -2 \le 0\\
&(5,2): -1 - 0 - 1 = -2 \le 0.
\end{aligned}
$$

Итак, мы получили клетку (4,1), которая не удовлетворяет условию оптимальности плана.

Путем перераспределения перевозок будем улучшать полученный план. Для этого выберем клетку с наибольшей разностью ($v_i-u_i-c_{ij}>0$). Начиная с этой клетки (4,1) построим цикл пересчета.

ОПРЕДЕЛЕНИЕ 2.3. Циклом пересчета в таблице транспортной задачи назовем ломаную линию, вершины которой находятся в заполненных клетках, в клетке пере-

счета эта линия имеет начало и конец, а звенья линии располагаются вдоль строк и столбцов таблицы.

Таблица 2.4

Карьер / Дорога	Карьер №1	Карьер №2	Карьер №3
№1	2 60	8	9
№2	3 70	5	8
№3	4 – 10	1 + 10	4
№4	2 +	4 – 70	7 60
№5	4	1	2 100

В цикл, образованный этой клеткой войдут клетки (3,1); (3,2); (4,2) (таблица 2.4). Начиная с выбранной клетки, (клетки пересчета) будем отмечать вершину ломаной линии знаками "+" и "–" по часовой стрелке. Причем клетки пересчета (4,1) и (3,2) со знаком "+" назовем "плюсовые клетки", а клетки (3,1) и (4,2) со знаком "–" – "минусовые клетки". В клетку пересчета записывают меньшее из чисел x_{ij}, стоящих в "минусовых" клетках. Одновременно это число прибавляют к числам, стоящим в "плюсовых" клетках и вычитают из чисел, стоящих в "минусовых" клетках. "Минусовая" клетка, содержащая это наименьшее число, освобождается (таблица 2.5).

Таблица 2.5

Карьер / Дорога	Карьер №1	Карьер №2	Карьер №3	
v_j	2	4	7	u_i
№ 1	2 60	8	9	0
№ 2	3 70	5	8	–1
№ 3	4	1 120	4	3
№4	2 10	4 60	7 60	0
№ 5	4	1	2 100	5

Клетка (3,1) стала свободной, в (4,1) записано 10, в (3,2) – 120 (110+10), а в (4,2) – 60 (70 – 10). Суммарные затраты на перевозку равны:

$2\cdot60 + 3\cdot70 + 1\cdot120 + 2\cdot10 + 4\cdot60 + 7\cdot60 + 2\cdot100 = 1330$.

Итак, затраты снизились на 1380 – 1330 = 50 единиц.

Вновь возвращаемся к нахождению потенциалов для нового плана. Для упрощения записи не будем выписывать систему уравнений, а все расчеты проведем в таблице, для чего введем в нее строку v_i и столбец u_i.

Записываем в столбец "u" в первой строке нуль (то есть $u_1=0$). Ищем в первой строке заполненные клетки и из выражения $v_i=c_{ij}+u_i$ находим $v_1=c_{11}+u_1=2+0=2$ (заполнена клетка (1,1)). Больше в первой строке нет заполненных клеток.

Переходим ко второй строке (заполнена клетка (2,1)). Для этой клетки в выражении $u_2=v_1-c_{21}$ известны v_1 и c_{21}, следовательно, $u_2=2-3=-1$. В третьей строке заполнена клетка (3,2), но для нее неизвестны потенциалы v_2 и u_3 , поэтому перейдем к следующей строке. Здесь есть три клетки (4,1), (4.2) и (4,3) из равенства $u_4=v_1-c_{41}=2-2=0$, найдем $u_4=0$. Тогда $v_2=u_4+c_{42}$ (для клетки (4,2)) равно 4, а $v_3=u_4+c_{43}=7$.

Теперь, возвращаясь к клетке (3,2), можно найти

$u_3=v_2-c_{32}=4-1=3$.

Наконец, для клетки (5,3) $u_5=v_3-c_{53}=5$. Проверим план на оптимальность.

(1,2): $4 - 0 - 8 = -4 \le 0$	(1,3): $7 - 0 - 9 = -2 \le 0$
(2, 2): $4 - 5 - (-1) = 0 \le 0$	(2,3): $7 - (-1) - 8 = 0 \le 0$
(3,1): $2 - 3 - 4 = -5 \le 0$	(3,3): $7 - 3 - 4 = 0 \le 0$
(5,1): $2 - 5 - 4 = -7 \le 0$	(5, 2): $4 - 1 - 5 = -2 \le 0$.

Итак, условие оптимальности опорного плана выполняется. Ненулевые компоненты оптимального опорного плана перевозок следующие:

$x_{11}=60$; $x_{21}=70$; $x_{41}=10$; $x_{32}=120$; $x_{42}=60$; $x_{43}=60$; $x_{53}=100$.

При этом плане получим минимальную стоимость перевозок, равную 1330 условных денежных единиц.

В заключение опишем алгоритм решения транспортной задачи методом потенциалов:

а) находим первоначальный "невырожденный" опорный план одним из известных методов;

б) вычисляем потенциалы u_i и v_j;
в) проверяем все незаполненные клетки на потенциальность (то есть выполнение условия $v_j - u_i - c_{ij} \le 0$). Если для всех клеток это неравенство справедливо, переходим к п. з), иначе к п. г);
г) среди положительных разностей ($v_j - u_i - c_{ij}$) ищем максимальную, клетка, содержащая эту максимальную разность - клетка пересчета;
д) отмечаем клетку пересчета знаком "+" и строим цикл пересчета, последовательно присваивая клеткам цикла знаки "–" и "+", начиная с клетки пересчета;
е) в клетках, помеченных знаком "–" находим наименьшую поставку и отнимаем ее от клеток "–", а к клеткам "+" прибавляем. При этом клетка, содержащая наименьшую поставку, освобождается;
ж) после получения нового распределения возвращаемся к п. б);
з) получен оптимальный план перевозок, конец решения.

2.3 Анализ чувствительности

Итоговое распределение перевозок, а также значения теневых цен, соответствующих пустым клеткам, позволяет проанализировать модель на чувствительность. А именно, выяснить, в каких пределах можно варьировать тарифы в незанятых клетках, чтобы это не привело к изменению оптимального плана перевозок. Поскольку теневая цена показывает, на какую максимальную (по модулю) величину можно уменьшить тариф на перевозку, чтобы включение клетки в план перевозок не привело бы к улучшению значения целевой функции. Например, в рассмотренной задаче теневая цена клетки (Д№1,К№2) равна 4–0–8=–4, из чего следует, что уменьшение тарифа в этой клетке в пределах от 8 до 4 не приведет к изменению оптимального плана.

Изменение тарифов в заполненных клетках в сторону уменьшения делает целесообразным увеличение плана в таких клетках, если это возможно. Если же тариф, стоящий в заполненной клетке, возрастает, то при

определенном его значении использование этой клетки становится нежелательным. Иллюстрируем это соображение примером.

Рассмотрим заполненную клетку (Д№1,К№1). Если тариф на перевозку станет больше "1", то обратим внимание на циклы, в которых задействована эта клетка. Имеется один такой цикл со свободной клеткой (Д№1,К№2). Теневая цена для (Д№1,К№1) – (–4). В указанном цикле клетка (Д№1,К№1) помечена знаком "–", и любое увеличение тарифа в этой клетке повлечет за собой снижение по модулю теневой цены свободной клетки. Изменение оптимального плана перевозок будет иметь место в том случае, если тариф для клетки (Д№1,К№1) возрастет на величину, большую чем модуль теневой цены свободной клетки, то есть на 4 единицы и превысит 6 единиц. При этом теневая цена клетки (Д№1,К№2) станет положительной и использование этой клетки в плане станет выгодным.

Таким образом, можно указать промежутки устойчивости оптимального плана по изменению тарифов для любых клеток – свободных и занятых.

Примечание. Если перевозка от какого-то поставщика какому–либо потребителю невозможна, то в алгоритме решения задачи это ограничение учитывается назначением достаточно большого тарифа.

3 Дискретное программирование

3.1 Задачи целочисленного линейного программирования

3.1.1 Задача о размещениях

Пусть имеются "m" пунктов, в которых могут размещаться предприятия, производящие некоторую единичную продукцию (станки, линии и т.д.). Кроме того заданы "n" пунктов потребления этой продукции с объемами потребления, равными соответственно b_1, b_2, ..., b_n, а также матрицы транспортных затрат $\boldsymbol{c} = \{c_{ij}\}_{mxn}$. Здесь c_{ij}– затраты на транспортировку единицы продукции из производящего пункта "i" в потребляющий пункт "j". Задача состоит в таком размещении предприятий, определении их производственных мощностей и организации перевозок, чтобы суммарные затраты по производству и транспортировке были минимальны. Обозначим через x_i – объем продукции в единицах, который необходимо производить в пункте "i", а через x_{ij}– количество единиц продукции, поставляемой из пункта "i" в пункт "j". Тогда затраты на производство продукции, если стоимость производства единицы продукции в пункте "i"–c_i, будут равны

$$\sum_{i=1}^{m} c_i x_i .$$

Затраты на транспортировку произведенной продукции к потребителям

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \ .$$

Вся задача формулируется так:

Найти такие x_i и x_{ij}, при которых

$$f = \sum_{i=1}^{m} c_i x_i + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \to \min$$

(суммарные затраты по производству и транспортировке минимальны), при условиях:

1) $x_i \geq 0,\ x_{ij} \geq 0$;

2) $x_i = \sum_{j=1}^{n} x_{ij}, \quad i = \overline{1,m}$ (производимая продукция полностью потребляется);

3) $\sum_{i=1}^{n} x_{ij} \geq b_j, j = \overline{1,n}$ (каждый потребитель получает продукции в объеме, не менее заданного значения);

4) $x_i,\ x_{ij}$ – принимают целочисленные значения.

3.1.2 Задача о назначениях

Пусть имеется "n" единиц оборудования различных типов, которое требуется распределить между "n" предприятиями, имеющими различный уровень технической оснащенности. Обозначим c_{ij} - производительность "i"-го типа оборудования на "j''-м предприятии. Задача состоит в таком распределении оборудования (по одному на предприятие), которое обеспечит максимальную производительность.

Определим неизвестные x_{ij} следующим образом:

$$x_{ij} = \begin{cases} 1, \text{ если ''i'' тип оборудования попадает на ''j'' предприятие;} \\ 0, \text{ в противном случае.} \end{cases}$$

Теперь задача может быть сформулирована следующим образом. Найти такие значения x_{ij}, чтобы производительность распределяемого оборудования была максимальной

$$P = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \to \max$$

при условиях:

1) $\sum_{i=1}^{n} x_{ij} = 1$, j=1,2,…,n (На каждое предприятие по одному виду оборудования),

2) $\sum_{j=1}^{n} x_{ij} = 1$, i = 1,2,…,n (Каждая единица оборудования распределяется на 1 предприятие).

Отметим, что возможна задача о назначениях, в которой целью является минимизация. Конкретный пример будет приведен при иллюстрации алгоритма решения.

3.1.3 Задача о коммивояжере

Имеется "n" городов. Выезжая из одного, коммивояжер должен объехать все и вернуться в исходный город. В каждый город можно заезжать только один раз. Требуется найти минимальный замкнутый маршрут.
Обозначим $s = \{s_{ij}\}_{nxn}$ - матрицу расстояний, x_{ij} - переменные, определим следующим образом:

$$x_{ij} = \begin{cases} 1, \text{если коммивояжер из города i перезжает в город j; j, i} = 1,2,..., n; j \neq i; \\ 0, \text{в противном случае.} \end{cases}$$

Требуется минимизировать

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \to \min$$

при условиях

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = \overline{1,n}$$

(из каждого города коммивояжер выезжает только 1раз);

$$\sum_{j=1}^{n} x_{ij} = 1, i = \overline{1,n}$$

(в каждый город коммивояжер въезжает 1 раз).

$$U_i - U_j + nx_{ij} \leq n - 1,\ i = \overline{1,n},\ j = \overline{1,n},\ i \neq j,$$

где U_i, U_j – произвольные действительные числа.

Последнее условие обеспечивает замкнутость маршрута и отсутствие петель.

3.2 Методы решения задач целочисленного программирования

3.2.1 Метод отсечения Гомори

Запишем общую задачу целочисленного программирования в виде: найти максимум

$$f(x)=\sum_{k=1}^{n} c_k x_k \qquad (3.1)$$

в области, определенной условиями

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad \mathrm{i}=\overline{1,\mathrm{m}} \qquad (3.2)$$

$$x_j > 0 \ , \quad \mathrm{j}=\overline{1,\mathrm{n}} \qquad (3.3)$$

$$x_{\mathrm{j}} - \text{целые}, \ \mathrm{j}=\overline{1,\mathrm{n}} \qquad (3.4)$$

Метод отсечения Гомори состоит в следующем:

а) решаем задачу линейного программирования (3.1) – (3.3);

б) полученное оптимальное решение задачи (3.1) – (3.3), если оно существует, проверяем на целочисленность:

– если все x_j, $\mathrm{j}=\overline{1,n}$ допустимые целые, то полученное оптимальное решение задачи линейного программирования является оптимальным решением задачи целочисленного программирования;

– если задача линейного программирования решения не имеет, то не имеет решения и задача целочисленного программирования;

– наконец, если хотя бы одна координата не удовлетворяет условию (3.4), то:

в) строим дополнительное линейное ограничение, с помощью которого отсекается та часть допустимой области, определяемой условиями (3.2) – (3.3), в которой

содержится оптимальное решение задачи линейного программирования (3.1) – (3.3), но нет ни одного допустимого решения, удовлетворяющего условию (3.4) и вновь выполняем пункт а) для задачи линейного программирования с дополнительным ограничением и т.д.

Гомори показал, что при "k-ом" возвращении к решению задачи линейного программирования, "k-ое" дополнительное ограничение имеет вид:

$$x_{n+k+1} = -(x_{i0} - [x_{i0}]) + \sum_{j \in N_k} (x_{ij} - [x_{ij}]) x_j \text{ , } k = 0,1,2,..., x_{n+k+1} \geq 0,$$

где $[x_{i0}]$, $[x_{ij}]$ – целая часть соответствующей величины;

x_{i0} – нецелая координата оптимального плана задачи (3.1) – (3.3) с наименьшим индексом;

x_{ij} – координаты разложения векторов A_j, не попавших в базис;

N_k – множество векторов, не попавших в базис.

Ниже приводится схема алгоритма метода Гомори (рисунок 3.1).

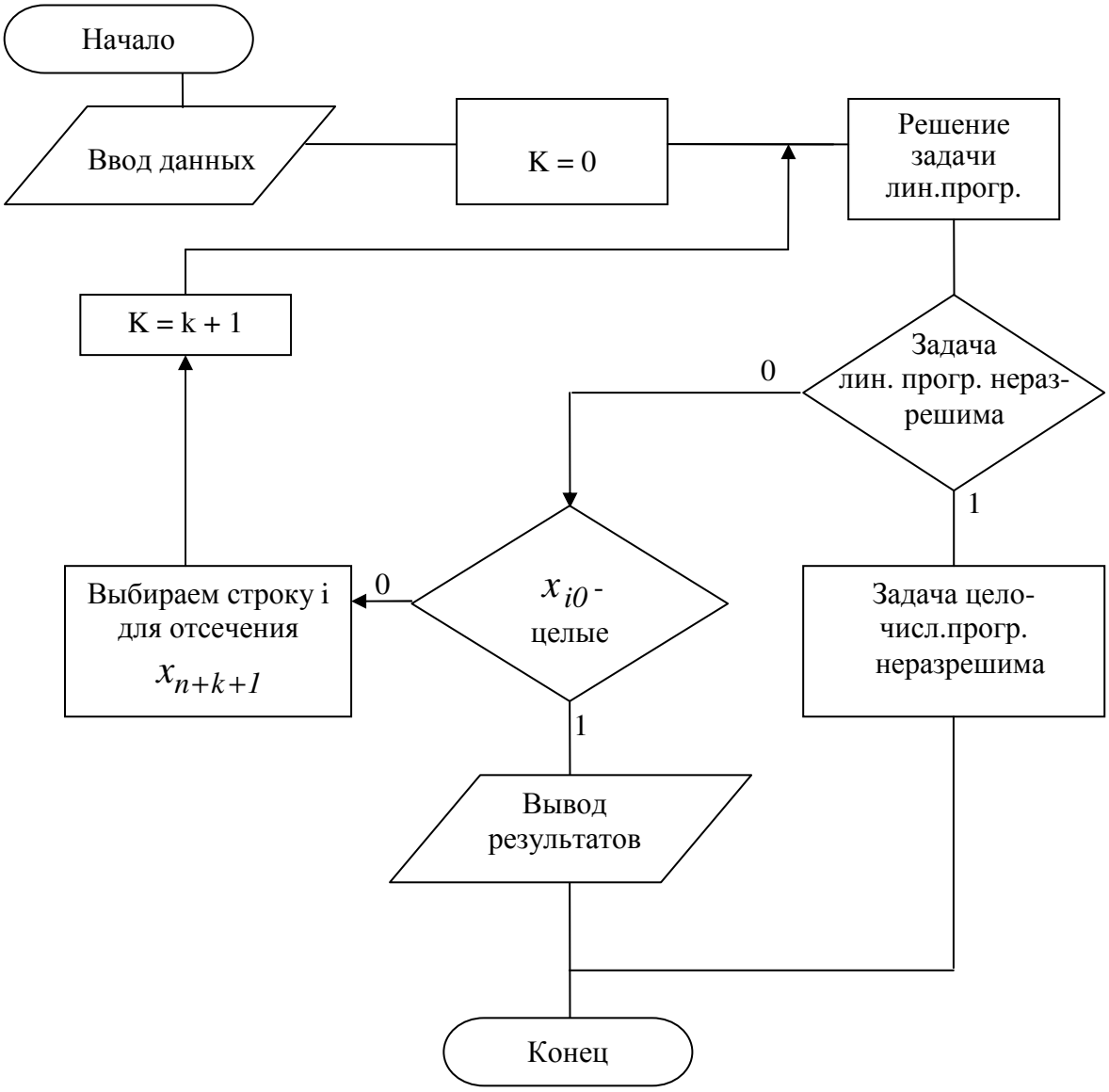


Рисунок 3.1 – Схема алгоритма метода Гомори

3.2.2 Метод ветвей и границ

У многих задач целочисленного программирования, в частности у сформулированных выше, множество всех допустимых решений представляет собой всевозможные комбинации (перестановки) одного и того же набора чисел. Естественно, что оптимальное решение можно найти, анализируя все возможные варианты. Вопрос лишь в том, как целесообразней организовать такой анализ. Существуют методы динамического программирования, метод ветвей и границ, использующие последовательный анализ вариантов.

Рассмотрим второй из них. Пусть G_0 – множество допустимых решений задачи, в которой среди совокупности $\boldsymbol{x} \in G_0$ требуется найти то $\boldsymbol{x}$, при котором целевая функция f имеет минимум. По некоторому закону (правилу) поставим в соответствие множеству G_0 число Z_0, которое является оценкой нижней границы целевой функции на множестве G_0. Разобьем множество G_0 на конечное число непересекающихся подмножеств (для наглядности на два) G_1 и G_2: $G_0=G_1 \cup G_2$, $G_1 \cap G_2=\varnothing$. Определим по выбранному закону (правилу) оценки нижней границы целевой функции на этих подмножествах Z_1 и Z_2.

Возьмем подмножество с меньшей оценкой, допустим G_2, и разобьем $G_2=G_3 \cup G_4$, $G_3 \cap G_4=\varnothing$ и найдем оценки подмножеств G_3 и G_4.

Подмножество с меньшей оценкой выбираем для ветвления и т.д. (рисунок 3.2).

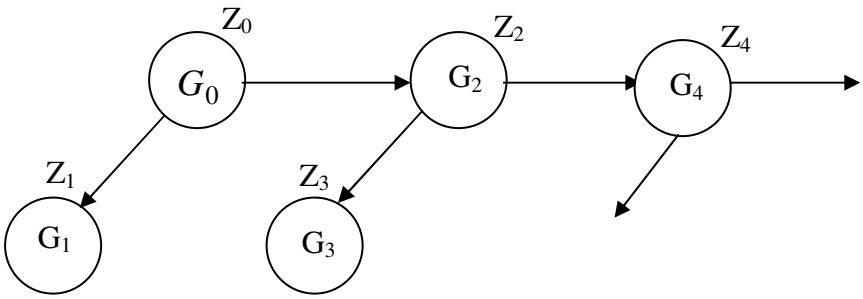


Рисунок 3.2

Естественно ожидать, что оптимальное решение с большей вероятностью содержится в подмножестве, оценка которого меньше оценок неветвленных подмножеств и, таким образом, производя ветвления, добраться до оптимального решения.

3.2.3 Метод ветвей и границ решения задачи о коммивояжере

Рассмотрим задачу о коммивояжере. Пусть $S=\{S_{ij}\}$; $i=\overline{1,n}$; $j=\overline{1,n}$ матрица, элемент которой S_{ij} определяет расстояние при переходе из пункта «i» в пункт «j». Полагаем $S_{ii}=\infty$, $i=\overline{1,n}$, рассматриваем это как запрет на переезд из "i" в "i".

ОПРЕДЕЛЕНИЕ 3.1. Циклом t назовем набор из "n" упорядоченных пар городов, образующих маршрут, который проходит через каждый город только один раз:

$t = \{(i_1, i_2), (i_2, i_3),\ldots, (i_{n-1}, i_n), (i_n, i_1)\}$.

ОПРЕДЕЛЕНИЕ 3.2. Издержками $Z(t)$ для цикла t назовем величину:

$$Z(t) = \sum_{k=1}^{n-1} S_{i_k, i_{k+1}} + S_{i_n, i_1}.$$

В нашей интерпретации $Z(t)$ – это длина замкнутого маршрута, образованного циклом t.

ОПРЕДЕЛЕНИЕ 3.3. Матрица, которая получается из данной вычитанием из элементов каждой строки минимального элемента этой строки, а затем вычитанием из элементов каждого столбца минимального элемента этого столбца, называется приведенной матрицей. Сумма вычитаемых в процессе приведения элементов называется приводящей константой.

Если обозначить $S_{i,j(i)}=\min\limits_{j=1,n} S_{i,j}$, $i=\overline{1,n}$ – минимальный элемент S в строке "i", тогда после вычитания этих элементов получится матрица S' с неотрицательными элементами

$$S'_{ij}=S_{ij}-S_{i,j(i)},\ \mathrm{i}=\overline{1,\mathrm{n}},\ \mathrm{j}=\overline{1,\mathrm{n}}.$$

Обозначим

$$S'_{i(j),j}=\min_{i=1,n} S'_{ij},\ \mathrm{j}=\overline{1,\mathrm{n}}$$

минимальный элемент S' в столбце j. Тогда после вычитания этих элементов получится приведенная матрица S'' с неотрицательными элементами

$$S''_{ij}=S'_{ij}-S'_{i(j),j},\ \mathrm{i}=\overline{1,\mathrm{n}};\ \mathrm{j}=\overline{1,\mathrm{n}}.$$

По определению приводящая константа

$$h=\sum_{i=1}^{n} S_{i,j(i)}+\sum_{j=1}^{n} S'_{i(j),j}.$$

Если $Z(t)$ – издержки цикла t для исходной матрицы, а $Z'(t)$–издержки цикла t после приведения, то $Z(t)=Z'(t)+h$ и, очевидно, что h является нижней границей издержек для всех циклов t исходной матрицы расстояний, поскольку h – сумма минимальных элементов строк и столбцов. Из этого следует, что применяя метод ветвей и границ, мы можем использовать h в качестве оценки разветвляемых подмножеств.

Определимся теперь с принципом разбиения на подмножества и нахождения оценок этих подмножеств. Пусть G_0 – множество всех маршрутов. Разобьем G_0 на два подмножества: первое подмножество состоит из всех маршрутов, включающих переезд из города "i" в город "j", говорят содержащих пару (i, j), а второе подмножество состоит из множества маршрутов не содержащих пару (i, j).

Пару городов (i, j) для ветвления будем выбирать среди тех пар, которым в приведенной матрице соответствуют нулевые элементы, причем выбирается такая пара (i, j), чтобы подмножество, не содержащее пару (i, j) имело максимальную оценку. Разветвляя последовательно, мы построим дерево ветвей, на вершине которого будет подмножество, содержащее две пары городов, завершающих маршрут. Спускаясь по дереву ветвей, мы по па-

рам городов, определяющих ветвление, определим все пары городов, составляющих маршрут (рисунок 3.3).

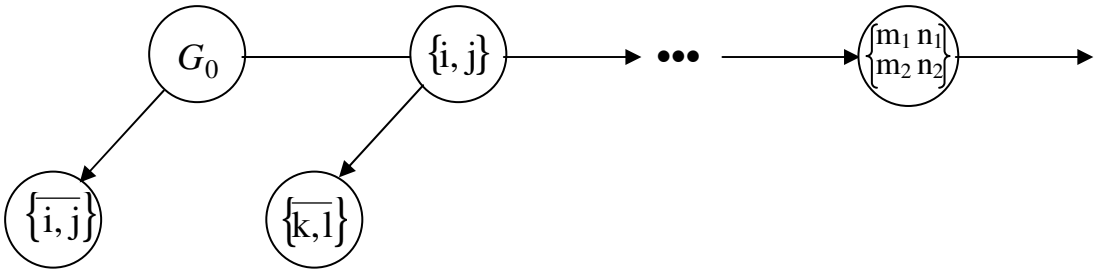


Рисунок 3.3 – Спуск по дереву ветвей

Опишем вычислительный алгоритм.

1. Осуществим приведение матрицы S по строкам и столбцам, получим приведенную матрицу S'.

2. Вычислим сумму приводящих констант $h^{(k)}$ - это оценка для исходного множества маршрутов G_0. Обозначим оценку $\omega(G_0) = h$.

3. Выберем претендентов для ветвления, т.е. те пары (i, j) i=l, 2, ..., j = 1, 2, ..., i ≠ j, для которых $S_{ij}^{(k)}=0$ и вычислим для всех таких $S_{ij}^{(k)}$ величины

$$\theta_{ij} = \min_{j' \neq j} S_{ij'} + \min_{i' \neq i} c_{ij}\ ,\ j' = 1,2,...,\ i' = 1,2...$$

4. Выберем для ветвления ту пару (i, j) из претендентов на ветвление, для которой θ_{ij} получится максимальным.

5. В качестве оценки множества всех маршрутов, не содержащих выбранную пару (i, j), возьмем оценку того множества, которое разветвляли плюс max θ_{ij}.

6. Так как из каждого города можно выезжать только один раз и в каждый город можно въезжать только один раз, то строку "i" и столбец "j" можно из дальнейшего рассмотрения исключить. Чтобы не получить замкнутых неполных циклов, нужно наложить необходимые запреты, в частности, на переезд из "j" в "i" то есть положить $S_{ij}=\infty$.

7. Если полученная после вычеркивания строки и столбца и наложения запретов матрица имеет размерность 2x2, то определяемые ею пары городов завершают

маршрут. Приводя эту матрицу и добавляя приводящую константу к оценке последнего разветвляемого множества, получим оценку маршрута.

Если эта оценка не больше оценки всех тупиковых ветвей, то маршрут, описываемый деревом ветвей, является оптимальным, иначе процесс ветвления должен быть продолжен, исходя из множества с меньшей оценкой.

8. Если усеченная матрица не имеет размерности 2x2, то приводим полученную матрицу и находим оценку множества {i, j}, то есть множества маршрутов, содержащих пару (i, j), как сумму приводящей константы и оценки разветвляемого множества.

Переходим к п. 3.

Пример 3.1. Имеется четыре пункта, расстояние между которыми описано матрицей расстояний. Найти оптимальный (минимальный) замкнутый маршрут объезда городов.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} \underline{1} & \underline{2} & \underline{3} & \underline{4} \end{array} \\ \begin{pmatrix} \infty & 13 & 12 & 4 \\ 13 & \infty & 7 & 8 \\ 12 & 7 & \infty & 5 \end{pmatrix} \end{array}$$

1. Приводим матрицу S:

$$\begin{array}{c} \begin{array}{cccc} \underline{1} & \underline{2} & \underline{3} & \underline{4} \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{pmatrix} \infty & 13 & 12 & 4 \\ 13 & \infty & 7 & 8 \\ 12 & 7 & \infty & 5 \\ 4 & 8 & 5 & \infty \end{pmatrix} \\ c_{i(j),\,j}\ \ 0\ \ 2\ \ 0\ \ 0 \end{array} \quad \begin{array}{c} c_{i,\,j(i)} \\ \begin{array}{c} 4 \\ 7 \\ 5 \\ 4 \end{array} \end{array} \quad \rightarrow \quad \begin{array}{c} \begin{array}{cccc} \underline{1} & \underline{2} & \underline{3} & \underline{4} \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{pmatrix} \infty & 9 & 8 & 0 \\ 6 & \infty & 0 & 1 \\ 7 & 2 & \infty & 0 \\ 0 & 4 & 1 & \infty \end{pmatrix} \end{array} \quad \rightarrow \quad \begin{array}{c} \begin{array}{cccc} \underline{1} & \underline{2} & \underline{3} & \underline{4} \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{pmatrix} \infty & 7 & 8 & 0 \\ 6 & \infty & 0 & 1 \\ 7 & 0 & \infty & 0 \\ 0 & 2 & 1 & \infty \end{pmatrix} \end{array}$$

2. Найдем сумму приводящих констант: h=4+7+5+4+2=22.

Найдем оценку множества G_0 : $\omega(G_0)$=22.

3. Укажем претендентов на ветвление и осуществим выбор пары для ветвления:

S_{14}=0, S_{23}= 0, S_{32}=0, S_{34}=0, S_{41}= 0,

θ_{14} =7+0=7, θ_{23}=1+1=2, θ_{32} = 0 +2=2, θ_{34}=0 +0=0,

$\theta_{41}= 6 + 1=7,$ $\max \theta_{ij}=\theta_{14} = 7$ (можно $\theta_{41} = 7$).

4. Итак, для ветвления выбираем пару (1,4) и начинаем строить дерево ветвей (рисунок 3.4). Найдем оценку множества {1,4}–множества всех маршрутов не содержащих пару (1,4): $\omega(\{1, 4\})=\omega(G_0)+\theta(1,4)=22+7=29$.

5. Вычеркнем строку "1", столбец "4" и положим запрет на переезд из "4" в"1":

$$\begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{ccc} \underline{1} & \underline{2} & \underline{3} \end{array} \\ \begin{pmatrix} 6 & \infty & 0 \\ 7 & 0 & \infty \\ \infty & 2 & 1 \end{pmatrix} \end{array}.$$

Приводим матрицу

$$\begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{ccc} \underline{1} & \underline{2} & \underline{3} \end{array} \\ \begin{pmatrix} 6 & \infty & 0 \\ 7 & 0 & \infty \\ \infty & 2 & 1 \end{pmatrix} \end{array}\begin{array}{c} c_{i,j(i)} \\ 0 \\ 0 \\ 1 \end{array} \rightarrow \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ c_{i(j),j} \end{array}\begin{array}{c} \begin{array}{ccc} \underline{1} & \underline{2} & \underline{3} \end{array} \\ \begin{pmatrix} 6 & \infty & 0 \\ \mathbf{7} & 0 & \infty \\ \infty & 1 & 0 \end{pmatrix} \\ \begin{array}{ccc} 6 & 0 & 0 \end{array} \end{array} \rightarrow \begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{ccc} \underline{1} & \underline{2} & \underline{3} \end{array} \\ \begin{pmatrix} 0 & \infty & 0 \\ 1 & 0 & \infty \\ \infty & 1 & 0 \end{pmatrix} \end{array}$$

6. Найдем $h = 1 + 6 = 7$ и оценку $\{1, 4\}$.

$$\omega(\{1, 4\})=\omega(G_0)+h=22+7=29.$$

7. Укажем претендентов для ветвления и выберем пару для ветвления:

$S_{21} = 0,$ $S_{23} = 0,$ $S_{32} = 0,$ $S_{43} = 0,$

$\theta_{21} = 0 + 1 = 1,$ $\theta_{23} = 0 + 0 = 0$, $\theta_{32} = 1+ 1 = 2,$
$\theta_{43}=1+0 = 1,$ $\max \theta_{ij} = \theta_{32} = 2.$

Итак, для ветвления берем пару (3,2).
Найдем оценку $\{3, 2\};$ $\omega(\{3, 2\})=\omega(\{1,4\}) + \theta_{32} = 29 + 2 = 31.$

8. Вычеркнем строку "3", столбец "2" и наложим запрет на переезд из "2" в "3".

$$\begin{array}{c} \\ 2 \\ 4 \end{array}\begin{array}{c} \begin{array}{cc} 1 & 3 \end{array} \\ \begin{bmatrix} 0 & \infty \\ \infty & 0 \end{bmatrix} \end{array}$$ – матрица приведенная, следовательно h=0 и

$$\omega(\{3, 2\})=\omega(\{1, 4\})+h=29.$$

9. Так как матрица размерности 2x2, то не запрещенные пары (2,1) и (4,3) завершают маршрут (рисунок 3.4).

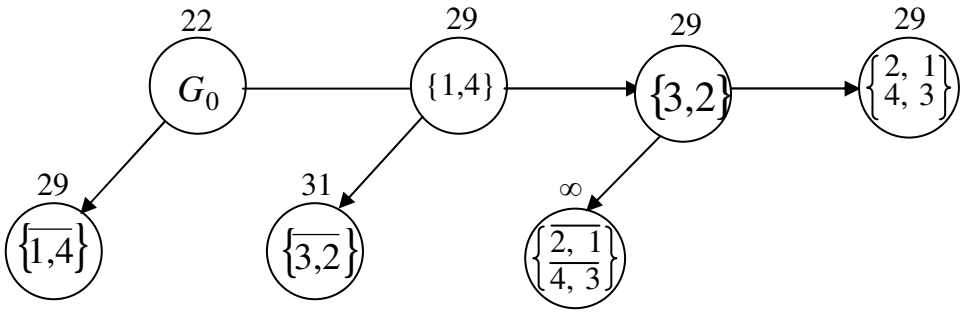


Рисунок 3.4 – Маршрут объезда пунктов

Поскольку оценка последнего подмножества не больше оценок тупиковых ветвей, то пары (1,4), (3,2), (2,1), (4,3) задают оптимальный маршрут, который можно представить, скажем так: 1→4 →3 →2 →1 с расстоянием S_{min}=29.

3.2.4 Аппроксимация решения задачи о коммивояжере

Приведенный в предыдущем пункте алгоритм является чрезвычайно трудоемким даже при использовании ЭВМ. В практически приемлемое время он позволяет решать лишь задачи небольшой размерности. В [5] предложено производить декомпозицию задачи большой размерности на ряд задач приемлемой размерности разбиением на группы (зоны) "близких" друг к другу объектов. В каждой группе выбираются объекты (вообще говоря разные), которые служат представителями группы (зоны) при определении минимальных расстояний между группами (зонами).

Используя описанный выше алгоритм находят, на первом этапе, оптимальный межгрупповой (межзональный) маршрут, а на втором этапе в каждой группе (зоне) ищут кратчайший маршрут между теми объектами группы (зоны), которые служат их представителями в межзональных расстояниях. Для того, чтобы построить минимальный незамкнутый маршрут между двумя представителями группы, с обязательным посещением всех остальных объектов группы, расстояния между этими двумя представителями переопределяют достаточно большими числами (накладывают "запрет" на ранний переезд) и затем этот переезд из оптимального внутригруппового маршрута выбрасывают. Полученный таким образом маршрут достаточно хорошо аппроксимирует искомый оптимальный маршрут при удачном разбиении на группы.

4 Методы безусловной оптимизации

ОПРЕДЕЛЕНИЕ 4.1. Будем говорить, что функция $f(x_1, x_2, \ldots, x_n) \equiv f(\boldsymbol{x})$ имеет локальный минимум (максимум) в точке $\boldsymbol{x}^0 = (x_1^{(0)}, \ldots, x_n^{(0)})$, если существует σ – окрестность $E_\sigma(\boldsymbol{x}^{(0)})$ точки $\boldsymbol{x}^{(0)}$ такая, что $f(\boldsymbol{x}^{(0)}) \le f(\boldsymbol{x})$ $(f(\boldsymbol{x}^{(0)}) \ge f(\boldsymbol{x}))$ для всех $\boldsymbol{x} \in E_\sigma(\boldsymbol{x}^{(0)})$. Если эти неравенства справедливы для всех $\boldsymbol{x}$ из области определения, то будем говорить, что $\boldsymbol{x}^{(0)}$ является точкой глобального минимума (максимума) в области определения функции. Поскольку задача определения максимума $f(x)$ эквивалентна задаче нахождения минимума – $f(x)$, то в дальнейшем говорим только о минимизации функции.

Классический подход к задаче нахождения $\boldsymbol{x}^{(0)}$ - точек локального минимума для дважды дифференцируемой функции состоит:

1) в определении таких точек $\boldsymbol{x}$, в которых все первые частные производные 1-го порядка обращаются в нуль (необходимое условие);

2) проверка положительной определенности матрицы Гессе $\boldsymbol{H}(\boldsymbol{x}) = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right) (i = \overline{1,n};\ j = \overline{1,n})$ в таких точках (достаточное условие).

Существенную трудность при таком подходе составляют как первый, так и второй этапы решения задачи, поэтому зачастую более предпочтительными являются численные методы.

4.1 Численные методы безусловной минимизации функции одной переменной. Методы прямого линейного поиска

Описываемые ниже методы минимизации требуют, чтобы минимизируемая функция обладала свойством квазивыпуклости.

ОПРЕДЕЛЕНИЕ 4.2. Будем говорить, что функция $f(x)$ квазивыпукла на отрезке $[a,b]$, если для всех x, принадлежащих $[x_1,x_2]$: $f(x)\le\max(f(x_1),f(x_2))$, при любых $x_1,x_2\in[a,b]$ (рисунок 4.1). Если неравенство строгое, то $f(x)$ строго квазивыпукла.

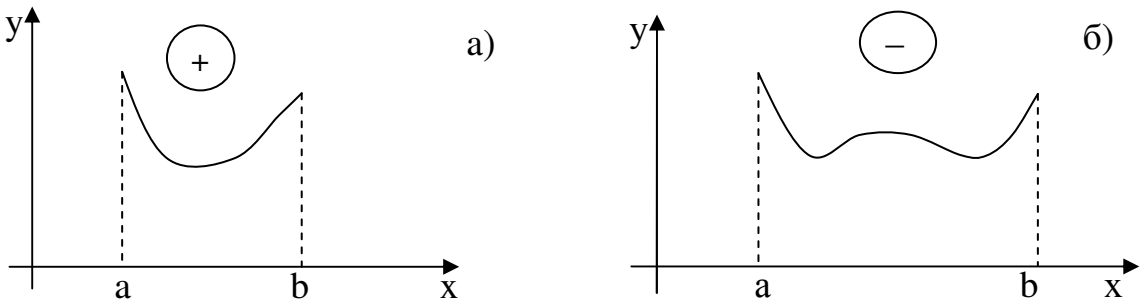


Рисунок 4.1 – Функции:
а) – квазивыпуклая; б) – неквазивыпуклая

Будем искать минимум функции $f(x)$ на некотором промежутке $[a,b]$. В основе предлагаемых ниже численных методов приближенного отыскания минимума лежит теорема.

ТЕОРЕМА 4.1. Пусть функция $f(x)$ строго квазивыпукла на $[a,b]$ и пусть c, d - любые точки принадлежащие $[a,b]$ такие, что $c<d$. Если $f(c)>f(d)$, то $f(x)\ge f(d)$ для всех $x\in[a,c]$. Если $f(c)\le f(d)$, то $f(x)\ge f(c)$ для всех $x\in[d,b]$ (рисунок 4.2).

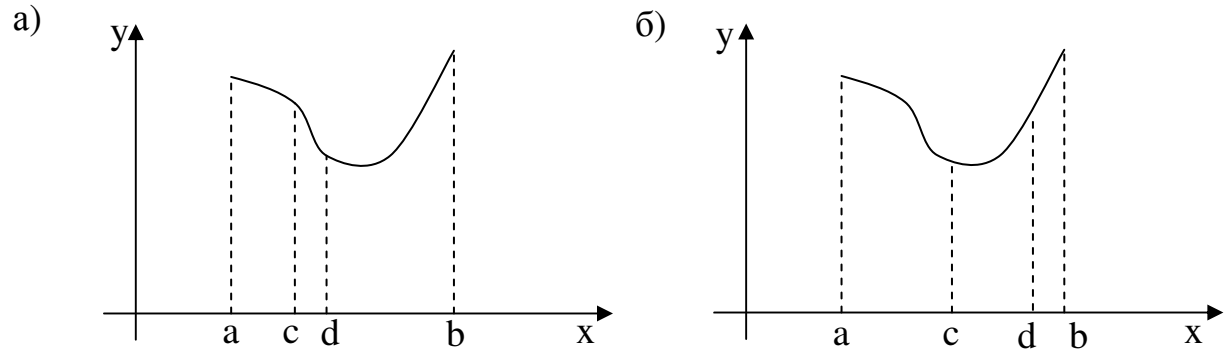


Рисунок 4.2 – Квазивыпуклая функция:
а) f(c)>f(d); б) f(c)≤f(d)

Из теоремы следует, что при $f(c)>f(d)$ точка минимума функции $f(x)$ может находиться только на отрезке $[c,b]$, а при $f(c)\le f(d)$ точка минимума может находиться только на $[a,d]$.

Сужая таким образом отрезок, на котором находится точка минимума, мы сможем сделать длину такого отрезка (интервала неопределенности) меньше требуемой точности ε и взять в качестве приближенного решения любую точку $x^{(0)}$ полученного промежутка.

4.1.1 Метод равномерного поиска

Пусть минимум функции $f(x)$ ищется на $[a,b]$. Покроем отрезок $[a,b]$ сеткой узлов

$$x_1=a,\ x_2=a+h,\ x_3=a+2h,\ldots,\ x_n=b=a+(n-1)h.$$

Вычислим значения $f(x)$ во всех узлах сетки. Выберем среди найденных значений минимальное. Допустим, минимальное значение функция принимает в узле x_k. Тогда, на основании теоремы, можем утверждать, что точка минимума $f(x)$ принадлежит интервалу $(x_k-h,\ x_k+h)$. Таким образом, после n вычислений функции интервал неопределенности сокращен до длины $2h$. Далее алгоритм уточнения точки минимума очевиден.

4.1.2 Метод золотого сечения

Естественно желание достичь интервала неопределенности требуемой длины при минимуме вычислений. Одним из наиболее эффективных в этом плане методов, является метод золотого сечения. Пусть минимум $f(x)$ ищется на $[a_1,b_1]$ и пусть точки c_1 и d_1 принадлежат $[a_1,b_1]$ и $c_1<d_1$. В силу теоремы 4.1 интервал неопределенности будет $[a_1,b_1]$, если $f(c_1)>f(d_1)$, и $[a_1,d_1]$, если $f(c_1)\le f(d_1)$. Новый интервал неопределенности обозначим $[a_2,b_2]$ и договоримся точки c_1, d_1, порождающие новый промежуток неопределенности $[a_2,b_2]$ искать так, чтобы:

1) длина нового интервала неопределенности b_2-a_2 не зависела от того, выполняется ли $f(c_1)>f(d_1)$ или $f(d_1)\geq f(c_1)$;

2) $b_1-c_1 = d_1-a_1$;

3) при новом делении отрезка $[a_2,b_2]$ точки деления c_2, d_2 выбирать так, что либо $c_2=d_1$, либо $d_2=c_1$, что позволяет сократить число вычислений функции $f(x)$ на каждом этапе (кроме исходного) до одного.

Рисунок 4.3 - Деление отрезка : а) $f(c_1)>f(d_1)$; б) $f(c_1)\leq f(d_1)$

а) здесь: $a_2=c_1$, $b_2=b_1$, $c_2=d_1$ и, как нетрудно показать, $d_2= c_2+0,382(b_2-c_2)$;

б) Здесь: $a_2= a_1$, $b_2= d_1$, $d_2= c_1$, $c_2= a_2+0,618(d_2-a_2)$.

Приведем целиком алгоритм метода золотого сечения.

Первый шаг. Выберем

$$c_1=a_1+0,382(b_1-a_1),\ d_1= a_1+0,618(b_1-a_1)$$

Положим $k=1$.

Второй шаг. Если $b_k-a_k<\varepsilon$, то оптимальная точка – это любая точка $[a_k,b_k]$. Иначе, если $f(c_k)>f(d_k)$, то переходим к третьему шагу, а если $f(c_k)\leq f(d_k)$, то к шагу 4.

Третий шаг. $a_{k+1}=c_k$, $b_{k+1}=b_k$, $c_{k+1}=d_k$,

$$d_{k+1}=a_{k+1}+0,618(b_{k+1}-a_{k+1}),$$

вычислим $f(d_{k+1})$, $k=k+1$ и переходим ко 2-му шагу.

Четвертый шаг. $a_{k+1}=a_k$,$b_{k+1}=d_k$, $d_{k+1}=c_k$,

$$c_{k+1}=a_{k+1}+0,318(b_{k+1}-a_{k+1}),$$

вычислим $f(c_{k+1})$, $k=k+1$ и переходим ко 2-му шагу.

Существенным препятствием для применения изложенных методов является требование квазивыпуклости минимизируемой функции. Во многих задачах это предположение не выполняется или не может быть легко проверено. Одним из путей преодоления этой трудности, в случае большого интервала неопределенности, является разделение его на маленькие интервалы, нахождение ми-

нимума на каждом из подинтервалов и последующий выбор наименьшего значения из минимумов на подинтервалах.

Из других методов, не требующих квазивыпуклости, отметим известные нам методы половинного деления, Ньютона, простых итераций, позволяющих найти решение уравнения $f(x)=0$, но требующих дополнительного исследования в окрестности найденных корней на предмет наличия минимума (максимума).

4.2 Численные методы безусловной минимизации функции многих переменных

Ставится задача минимизации функции $f(\boldsymbol{x}) = f(x_1, x_2, ...x_n)$ в некоторой замкнутой области. Пусть $\boldsymbol{a} = (a_1, a_2, ..., a_n)$ точка минимума $f(\boldsymbol{x})$. Будем говорить, что с точностью до ε точка $\boldsymbol{x}$ может быть взята в качестве приближенного значения точки минимума, если

$\| \boldsymbol{a} - \boldsymbol{x} \| < \varepsilon$, где $\| \boldsymbol{a} - \boldsymbol{x} \| = \sqrt{\sum_{i=1}^{n} (a_i - x_i)^2}$ или

$\| \boldsymbol{a} - \boldsymbol{x} \| = max \left| a_i - x_i \right| \quad (i = \overline{1,n})$.

Для геометрической иллюстрации методов будем использовать функцию двух переменных. Напомним, что линиями уровня функции $Z=f(x,y)$ называют множество точек (x,y), удовлетворяющих уравнению $f(x,y)=c$. Меняя c, мы получаем различные линии уровня функции $f(x,y)$. Геометрически линия уровня – это проекция на плоскость XOY линии пересечения $Z=f(x,y)$ и плоскости $Z=c$. Имея множество линий уровня, мы получаем представление о поведении $Z=f(x,y)$, говорят о рельефе функции $Z=f(x,y)$.

4.2.1 Методы многомерного прямого поиска

Суть методов многомерного прямого поиска, изложенных ниже, в том, что выбирают некоторую точку $\boldsymbol{x}^{(0)} = (x_1^{(0)}, ..., x_n^{(0)})$ и допустимое направление поиска $\boldsymbol{d}$.

Затем, отправляясь от точки $\boldsymbol{x}^{(0)}$ в направлении $\boldsymbol{d}$, минимизируют функцию одного переменного λ: $f(\boldsymbol{x}+\lambda\boldsymbol{d})$, изложенными выше методами.

Найдя $\lambda^{(0)}$, при котором $f(\boldsymbol{x}^{(0)}+\lambda\boldsymbol{d})$ получает минимальное значение, мы тем самым нашли точку $\boldsymbol{x}^{(1)}=\boldsymbol{x}^{(0)}+\lambda^{(0)}\boldsymbol{d}$, в которой значение *f*, вообще говоря, меньше чем в точке $\boldsymbol{x}^{(0)}$. Далее, отправляясь от $\boldsymbol{x}^{(1)}$ в некотором новом направлении $\boldsymbol{d}_1$, получаем некоторую точку $\boldsymbol{x}^{(2)}$, в которой значение *f* вообще говоря, меньше чем в $\boldsymbol{x}^{(1)}$ и т.д. Возникают вопросы: 1) Как целесообразнее выбирать $\boldsymbol{d}_i$? 2) Сходится ли последовательность $\boldsymbol{x}^{(0)}$, $\boldsymbol{x}^{(1)}$, . . .? 3) Как оценить погрешность?

4.2.2 Метод циклического покоординатного спуска

Опишем алгоритм одного из таких методов, выбирая в качестве направлений поиска координатные векторы $\boldsymbol{d}_1=\boldsymbol{l}_1,\boldsymbol{d}_2=\boldsymbol{l}_2,...,\boldsymbol{d}_n=\boldsymbol{l}_n$, т.е. $\boldsymbol{d}_i=\boldsymbol{l}_i$ - вектор, все компоненты которого, за исключением *i* равны нулю.

Тогда

$$\boldsymbol{x}^{(0)}+\lambda\boldsymbol{l}_i=\left(x_1^{(0)},x_2^{(0)},...,x_{i-1}^{(0)},x_i^{(0)}+\lambda,x_{i+1}^{(0)},...,x_n^{(0)}\right),$$

$$f(\boldsymbol{x}^{(0)}+\lambda\boldsymbol{l}_i)=f\left(x_1^{(0)},x_2^{(0)},...,x_{i-1}^{(0)},x_i^{(0)}+\lambda,x_{i+1}^{(0)},...,x_n^{(0)}\right)$$ и, следовательно, при минимизации функции $f(\boldsymbol{x}^{(0)}+\lambda\boldsymbol{l}_i)$, речь идет о минимизации функции одной переменной $x_i=x_i^{(0)}+\lambda_i$, при фиксированных остальных переменных.

Выберем начальную точку $\mathbf{y}_1=\boldsymbol{x}^{(0)}$, направление $\boldsymbol{d}_1=\boldsymbol{l}_1$ и, минимизируя $f(\mathbf{y}_1+\lambda\boldsymbol{l}_1)$, найдем, что минимум этой функции достигается при λ_1, в точке $\mathbf{y}_2=\mathbf{y}_1+\lambda_1\boldsymbol{l}_1=\left(x_1^{(0)}+\lambda_1,x_2^{(0)},...,x_n^{(0)}\right)$. Положим $\mathbf{y}_2=\mathbf{y}_1+\lambda_1\boldsymbol{l}_1$, выберем направление $\boldsymbol{l}_2$ и, минимизируя $f(\mathbf{y}_2+\lambda\boldsymbol{l}_2)$ най-

дем, что минимум этой функции достигается при λ_2 в точке $\mathbf{y}_3 = \mathbf{y}_2 + \lambda_2 \boldsymbol{l}_2 = \left(x_1^{(0)} + \lambda_1, x_2^{(0)} + \lambda_2, x_3^{(0)}, ..., x_n^{(0)} \right)$.

После n шагов найдем

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \lambda_n \boldsymbol{l}_n = \left(x_1^{(0)} + \lambda_1, x_2^{(0)} + \lambda_2, ..., x_n^{(0)} + \lambda_n \right).$$

Положим $\boldsymbol{x}^{(1)} = \mathbf{y}_{n+1}$ и тем самым завершим один цикл покоординатного спуска.

Отправляясь от $\boldsymbol{x}^{(1)}$, как от начальной точки, можем найти $\boldsymbol{x}^{(2)}$ и т.д.

Процесс завершить, если

$$\| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \| < \varepsilon,$$

где ε – требуемая точность,

$$\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = \sqrt{\sum_{i=1}^{n} \left(x_i^{(k+1)} - x_i^{(k)} \right)^2} \text{ или}$$

$$\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = \max | x_i^{(k+1)} - x_i^{(k)} | \, (i = \overline{1,n}).$$

Проиллюстрируем метод с помощью функции двух переменных (рисунок 4.4).

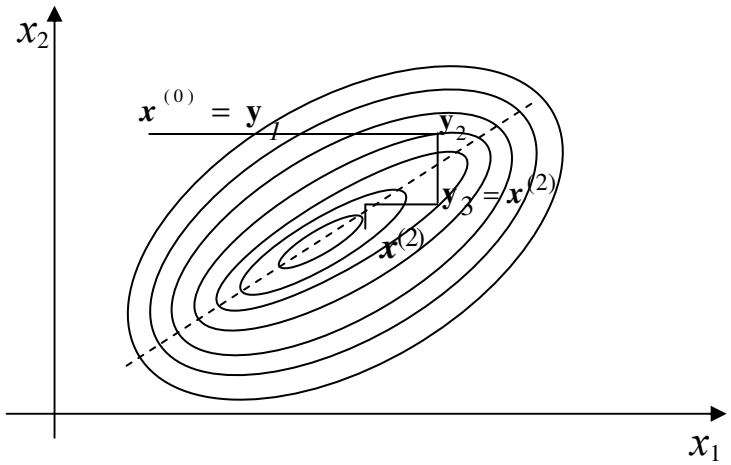


Рисунок 4.4 – Метод циклического покоординатного спуска

Двигаясь от точки $\boldsymbol{x}^{(0)} = \mathbf{y}_1$ параллельно оси OX_1, мы переходим от линий уровня с большим значением $f(x_1, x_2) = c$ к меньшим. Точка $\mathbf{y}_2$, в которой мы коснемся некоторой линии уровня, является точкой минимума

функции $f(x_1, x_2^{(0)})$ в направлении, параллельном оси OX_1. Двигаясь от точки $\mathbf{y}_2$ параллельно оси OX_2, мы, переходя от линий уровня с большим "c" к линиям уровня с меньшим "c", достигнем минимального значения в точке $\mathbf{y}_3$ - точке касания некоторой линии уровня. В точке $\mathbf{y}_3$ завершается цикл покоординатного спуска и получаем $\boldsymbol{x}^{(1)} = \mathbf{y}_3$.

Отправляясь от $\boldsymbol{x}^{(1)}$, как от начальной точки, можем найти $\boldsymbol{x}^{(2)}$ и т.д. Остается указать условия сходимости последовательности $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}$, ... и указать оценку погрешности.

Для сходимости метода циклического покоординатного спуска достаточно следующих требований:

1) минимум $f(\boldsymbol{x})$ вдоль любого направления единственен;

2) последовательность точек $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(k)}$ принадлежит некоторому замкнутому ограниченному подмножеству области $\boldsymbol{D}$.

Что касается погрешности, то ее можно определить по формуле: $\| \boldsymbol{a} - \boldsymbol{x}^{(k+1)} \| \le \| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \|$.

ЗАМЕЧАНИЕ. Если функция f не является дифференцируемой в некоторых точках, то метод может остановиться в неоптимальной точке (рисунок 4.5).

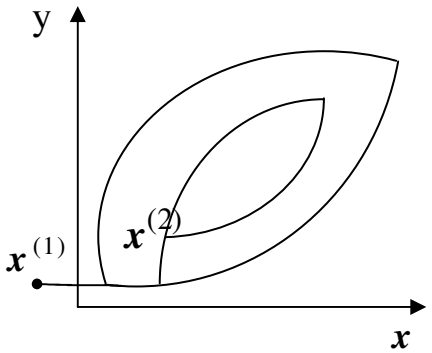


Рисунок 4.5 – Неоптимальная точка

Ясно, что поиск вдоль любой координатной оси в точке $\boldsymbol{x}^{(2)}$ не приводит к улучшению целевой функции. Это вызвано наличием так называемого оврага, то есть точки

недифференцируемой функции $f(\boldsymbol{x})$. Разрешить эту ситуацию можно используя, например, метод Хука–Дживса.

4.2.3 Метод Хука-Дживса

Метод Хука-Дживса осуществляет два типа поиска: исследующий поиск и поиск по образцу. Выбрав начальную точку $\boldsymbol{x}_1$ методом циклического покоординатного спуска, находим точку $\boldsymbol{x}_2$, т.е. осуществляем исследующий поиск. Если $\| \boldsymbol{x}_2 - \boldsymbol{x}_1 \| < \varepsilon$, то точка минимума найдена, иначе осуществляем поиск по образцу в направлении $\boldsymbol{x}_2 - \boldsymbol{x}_1 = \boldsymbol{d}$, что приводит нас в некоторую точку $\mathbf{y}$. Приняв $\mathbf{y}$ за $\boldsymbol{x}_1$ вновь проводим исследующий поиск и т.д. Сходимость метода Хука-Дживса обеспечена при тех же условиях, что и покоординатного спуска. Иллюстрация метода на рисунке 4.6 .

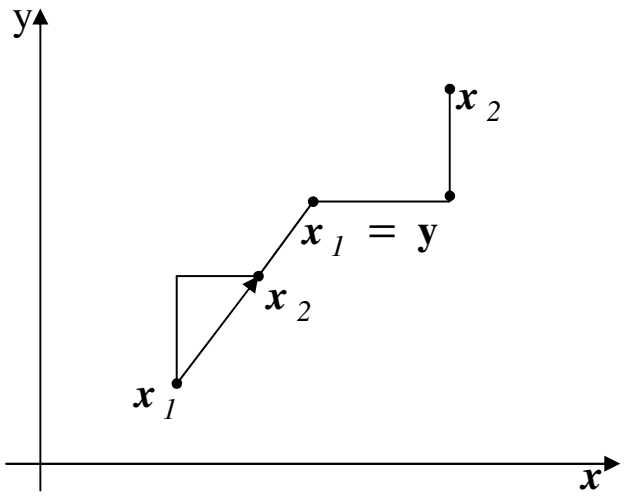


Рисунок 4.6 – Метод Хука-Дживса

4.2.4 Метод наискорейшего спуска

Известно, что направлением наибольшего возрастания функции f в точке $\boldsymbol{x}^{(0)}$ является направление, задаваемое вектором $grad\ f(\boldsymbol{x}^{(0)})$, а $(-grad\ f(\boldsymbol{x}^{(0)}))$ задает направление наибольшего убывания функции f в точке $\boldsymbol{x}^{(0)}$. Учитывая это, следует осуществлять линейный поиск не в направлении осей координат, а в направлении $-grad\ f$.

Рассмотрим алгоритм метода.

1. Выбираем $\boldsymbol{x}_i$, $i=1$.

2. Если $\| grad\ f(\boldsymbol{x}_i)\| < \varepsilon$, то $\boldsymbol{x}_i$ - искомая точка.

3. В противном случае положим $d_i = -grad\ f(\boldsymbol{x}_i)$ и решив задачу о минимуме $f(\boldsymbol{x}_i + \lambda d_i)$, найдем $\lambda_i > 0$.

4. Найдем $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \lambda_i d_i$ и переходим к пункту 2, положив $i = i+1$.

Сходимость метода обеспечена, если f непрерывно дифференцируема, а генерируемая последовательность, принадлежит замкнутому ограниченному множеству.

Недостатком метода наискорейшего спуска является зачастую медленная сходимость в окрестности стационарной точки. Это очевидно, например, в тех случаях, когда линии уровня вытянуты в окрестности оптимальной точки. О функциях, поверхности уровня которых сильно вытянуты, говорят, что она имеет "овражный" характер. Геометрически медленная сходимость объясняется так: спустившись на "дно оврага" мы, двигаясь в направлении $(-grad\ f)$, будем переходить с одного склона оврага на другой, то есть зигзагообразно продвигаться к точке минимума (рисунок 4.7).

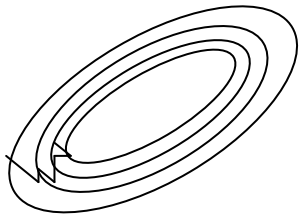


Рисунок 4.7 – Овражная функция

Что касается степени "овражности", то ее можно охарактеризовать с помощью минимального (λ_{min}) и максимального (λ_{max}) собственных чисел матрицы Гессе $-\left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)(i=\overline{1,n};\ j=\overline{1,n})$. Чем меньше λ_{min} / λ_{max}, тем больше овражность. Более предпочтительными в таких случа-

ях являются методы Ньютона и сопряженных направлений.

Коротко суть первого из них состоит в том, что функция $f(\boldsymbol{x})$ аппроксимируется (с помощью формулы Тейлора) многочленами второй степени, для которых находятся точки минимума. Последовательность таких точек приводит при определенных условиях к искомой точке минимума $f(\boldsymbol{x})$. Неудобством метода является необходимость многократного обращения матрицы Гессе.

Второй из методов для получения очередной точки требует проведения последовательной минимизации по каждому из n, специальным образом построенных направлений, которые называют сопряженными направлениями. Для квадратичной функции минимизация вдоль "n" таких направлений позволяет "точно" достичь точки минимума, а следовательно можно ожидать хороших результатов и для достаточно гладких функций. Подробнее с этими методами можно познакомиться в [4].

5 Задачи нелинейного программирования

5.1 Постановка задачи и основные определения

Выше рассматривались задачи линейного программирования. Это задачи, в которых все фигурирующие функции линейные. Однако большинство реальных задач не может быть адекватно описано с помощью моделей линейного программирования из-за нелинейности целевой функции или некоторых ограничений. В частности, в виде моделей нелинейного программирования можно представить задачи оптимизации, возникающие в следующих областях:

1) оптимальное управление;
2) проектирование строительных конструкций;
3) проектирование механических конструкций;
4) электрические цепи;
5) управление водными ресурсами;
6) распределение ресурсов в условиях неполной информации;
7) размещение оборудования.

Общая задача нелинейного программирования имеет вид:
минимизировать (максимизировать) целевую функцию

$$f(x_1, x_2, \ldots, x_n) \qquad (5.1)$$

при условиях

$$\begin{cases} g_i(x_1,x_2,...,x_n) \le b_i (i=\overline{1,k}) \\ g_i(x_1,x_2,...,x_n) = b_i (i=\overline{k+1,m}) \end{cases}, \qquad (5.2)$$

где f и g_i – некоторые известные функции n переменных, а b_i – заданные числа.

В результате решения задачи (1), (2) должна быть определена точка

$$\boldsymbol{x}^* = (x_1^*, x_2^*, ..., x_n^*),$$

координаты которой удовлетворяют соотношениям (2) и такая, что для всякой другой точки $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, удовлетворяющей условиям (2), выполняется неравенство

$$f(x_1^*, x_2^*, ..., x_n^*) \le f(x_1, x_2, ..., x_n) \text{ или}$$
$$[f(x_1^*, x_2^*, ..., x_n^*) \ge f(x_1, x_2, ..., x_n)].$$

Соотношения (2) могут включать и условия неотрицательности переменных. Если все функции f и g_i – линейные, то задача (1), (2) называется задачей линейного программирования. В противном случае имеем задачу нелинейного программирования.

Задачу (1), (2) можно записать сокращенно в векторной форме:
минимизировать (максимизировать) целевую функцию

$$f(\boldsymbol{x}) \qquad (5.3)$$

при условиях

$$\begin{cases} g_i(x) \le b_i \\ g_i(x) = b_i \end{cases} . \qquad (5.4)$$

Здесь f и g_i – определенные в евклидовом пространстве E_n функции. Совокупность всех векторов размерности n образует n-мерное евклидово пространство E_n.

В пространстве E_n система ограничений (2) определяет область допустимых решений задачи. В отличие от задачи линейного программирования она всегда является выпуклой.

Если определена область допустимых решений, то нахождение решения задачи (1), (2) сводится к определению такой точки этой области, через которую проходит гиперповерхность наинизшего (наивысшего) уровня: $f(x_1, x_2, ..., x_n) = c$. Указанная точка может находиться как на границе области допустимых решений, так и внутри нее.

5.2 Геометрическая интерпретация решения задач нелинейного программирования

Процесс нахождения решения задачи нелинейного программирования (1), (2) геометрическим способом состоит из следующих этапов:

1) нахождение области допустимых решений задачи, определяемой соотношениями (2) (если она пуста, то задача не имеет решения);

2) построение гиперповерхности $f(x_1, x_2, ..., x_n) = c$;

3) определение гиперповерхности наинизшего (наивысшего) уровня или установление неразрешимости задачи из-за неограниченности функции (1) снизу (сверху) на множестве допустимых решений;

4) нахождение точки области допустимых решений, через которую проходит гиперповерхность наинизшего (наивысшего) уровня и нахождение значения функции (1) в этой точке.

Пример 5.1.

Минимизировать функцию $(x_1 - 3)^2 + (x_2 - 2)^2$

при условиях

$$\begin{cases} x_1^2 - x_2 - 3 \le 0 \\ x_2 - 1 \le 0 \\ -x_1 \le 0. \end{cases}$$

Целевая функция и три ограничения имеют вид

$$f(x) = (x_1 - 3)^2 + (x_2 - 2)^2$$
$$g_1(x) = x_1^2 - x_2 - 3$$
$$g_2(x) = x_2 - 1$$
$$g_3(x) = -x_1.$$

Здесь функции $f(x)$ и $g_1(x)$ нелинейные, поэтому имеем задачу нелинейного программирования.

На рисунке 5.1 изображена допустимая область.

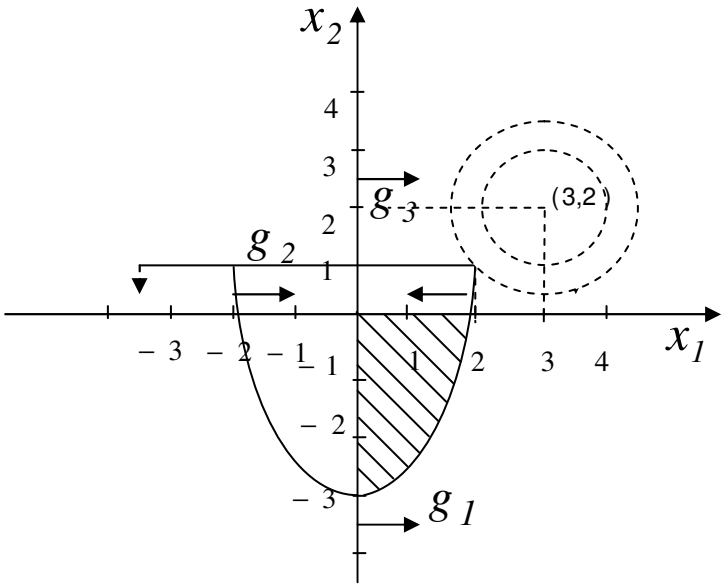


Рисунок 5.1 – Допустимая область для примера 5.1

Задача заключается в нахождении такой точки из допустимой области, для которой $(x_1-3)^2+(x_2-2)^2$ имеет наименьшее возможное значение. Заметим, что точки (x_1,x_2), удовлетворяющие равенству $(x_1-3)^2+(x_2-2)^2=c$, лежат на окружности радиуса $\sqrt{c}$ с центром в точке (3,2). Для каждого неотрицательного c такая окружность называется линией уровня целевой функции, отвечающей заданному значению c. Таким образом, задача заключается в нахождении минимального c, при котором хотя бы одна точка окружности принадлежит допустимой области. Как видно на рисунке 5.1, такая окружность наименьшего радиуса соответствует $c=2$ и пересекает допустимую область в единственной точке $A(2,1)$. Поэтому точка $A(2,1)$ - оптимальное решение, и значение целевой функции в этой точке равно 2.

Пример 5.2.

Максимизировать $6x_1-x_1^2+x_2$

при условиях

$$\begin{cases} 2x_1+3x_2\le 24 \\ x_1+2x_2\le 15 \\ 3x_1+2x_2\le 24 \\ x_2\le 4 \end{cases}$$
$$x_1,x_2\ge 0.$$

На рисунке 5.2 изображена допустимая область задачи в виде многоугольника $OABC$. Следовательно, для нахождения ее решения нужно определить такую точку многоугольника $OABC$, в которой целевая функция $6x_1 - x_1^2 + x_2$ принимает максимальное значение.

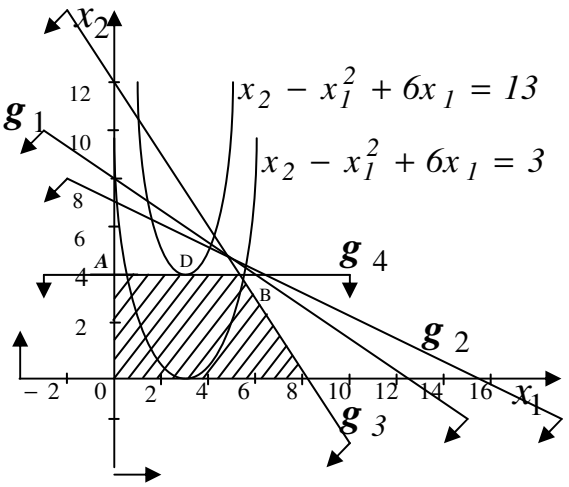


Рисунок 5.2 – Допустимая область для примера 5.2

Построим линию уровня $x_2 - x_1^2 + 6x_1 = c$ и исследуем ее поведение при различных значениях c. При каждом значении c получаем параболу, которая тем выше отдалена от оси Ox_1, чем больше значение c. Целевая функция принимает максимальное значение в точке касания одной из парабол с границей многоугольника $OABC$. В данном случае это точка D, в которой линия уровня $x_2 - x_1^2 + 6x_1 = 13$ касается допустимой области. Координаты точки D найдем из системы уравнений:

$$\begin{cases} x_2 - x_1^2 + 6x_1 = 13 \\ x_2 = 4. \end{cases}$$

Точка D (3,4) – оптимальное решение и значение целевой функции в этой точке равно 13.

Как видно из примера 5.2, точка максимального значения целевой функции не является вершиной допустимой области. Поэтому процедура перебора вершин, используемая при решении задачи линейного программирования, не применима для решения данной задачи.

Очевидно, что такой геометрический способ решения подходит для решения очень простых задач и практиче-

ски неприменим в задачах, где число переменных больше двух.

5.3 Задачи выпуклого программирования

Рассмотрим задачу нелинейного программирования.

Минимизировать (максимизировать)

$$f(x_1, x_2, ..., x_n) \tag{5.5}$$

при условиях

$$\begin{cases} g_i(x_1, x_2, ..., x_n) \le b_i \left(i = \overline{1,m}\right) \\ x_j \ge 0 \left(j = \overline{1,n}\right) . \end{cases} \tag{5.6}$$

Для решения этой задачи в такой общей постановке не существует универсальных методов. Однако для отдельных классов задач, в которых имеются дополнительные ограничения относительно свойств функций f и g_i, разработаны эффективные методы их решения. В частности, ряд таких методов имеется для решения задач при условии, что f – выпуклая (вогнутая) функция и область допустимых решений, определяемая (5.6) и (5.7) – выпуклая.

Дадим определения и понятия, необходимые нам в дальнейшем.

5.3.1 Основные определения и теоремы

ОПРЕДЕЛЕНИЕ 5.1. Непустое множество S из E_n называется выпуклым, если отрезок прямой, соединяющий две любые точки множества S также принадлежит этому множеству. Иными словами, если точки $\boldsymbol{x}_1$ и $\boldsymbol{x}_2$ лежат в S, то точка $\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2$ также должна принадлежать S для всех $\lambda \in [0, 1]$.

На рисунке 5.3 приведены примеры множеств.

Рисунок 5.3 – Примеры множеств:
а) – выпуклое; б) – невыпуклое

ОПРЕДЕЛЕНИЕ 5.2. Пусть $f : S \to E_1$, где S – непустое выпуклое множество в E_n Говорят, что функция f выпукла на S, если

$$f[\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2] \le \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2) \qquad (5.7)$$

для любых $\boldsymbol{x}_1$, $\boldsymbol{x}_2 \in S$ и $\lambda \in [0, 1]$.

При этом, если имеет место строгое неравенство, то говорят о строгой выпуклости.

Функция f называется вогнутой, если $-f$ выпукла на S.

На рисунке 5.4 приведены примеры выпуклых и вогнутых функций.

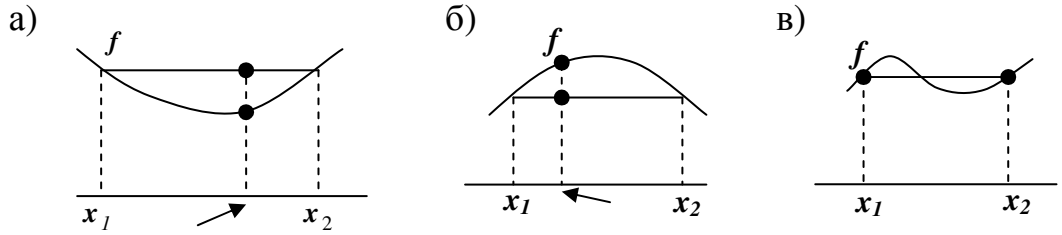


Рисунок 5.4 – Функции : а) – выпуклая функция; б) – вогнутая функция; в) – функция, не являющаяся ни вогнутой, ни выпуклой.

Замечания. 1.Из определения квазивыпуклой функции в п.4.1 очевидно следует, что выпуклая функция является также и квазивыпуклой, причем строго квазивыпуклой. Таким образом,предположение о квазивыпуклости исследуемой функции является существенно более слабым, чем выпуклость этой функции.

2. Строго квазивыпуклые (квазивогнутые) функции особенно важны в нелинейном программировании,т.к. для этих функций локальный минимум (максимум) на

выпуклом множестве соответственно являются глобальными минимумом и максимумом.

ОПРЕДЕЛЕНИЕ 5.3. Говорят, что множество допустимых решений задачи (5)–(6) удовлетворяет условию регулярности, если существует по крайней мере одна точка $\boldsymbol{x}$, принадлежащая области допустимых решений такая, что

$$g_i(\boldsymbol{x}) < b_i,\ (i = \overline{1,m}).$$

ОПРЕДЕЛЕНИЕ 5.4. Задача (5)–(6) называется задачей выпуклого программирования, если функция f является вогнутой (выпуклой), а функции g_i – выпуклыми.

ОПРЕДЕЛЕНИЕ 5.5. Пусть $\boldsymbol{x}^* \in E_n$. Если $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ для всех $\boldsymbol{x}$ из E_n, то $\boldsymbol{x}^*$ называется точкой глобального минимума функции f. Если существует ε – окрестность $N_\varepsilon(\boldsymbol{x}^*)$ точки $\boldsymbol{x}^*$ такая, что $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ для всех $\boldsymbol{x} \in N_\varepsilon(\boldsymbol{x}^*)$, то $\boldsymbol{x}^*$ называется точкой локального минимума. При обратных неравенствах соответственно имеют место точки глобального максимума и локального максимума.

ОПРЕДЕЛЕНИЕ 5.6. Функцией Лагранжа задачи выпуклого программирования называется функция

$$L(x_1, x_2, ..., x_n, \lambda_1, \lambda_2, ..., \lambda_m) = \\ = f(x_1, x_2, ..., x_n) + \sum_{i=1}^{m} \lambda_i [b_i - g_i(x_1, x_2, ..., x_n)], \qquad (5.8)$$

где $\lambda_1, \lambda_2, ..., \lambda_m$ - неопределенные множители Лагранжа.

ОПРЕДЕЛЕНИЕ 5.7. Точка $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0) = (x_{10}, x_{20}, ..., x_{n0}, \lambda_{10}, \lambda_{20}, ..., \lambda_{m0})$ называется седловой точкой функции Лагранжа, если

$$L(x_1, x_2, ..., x_n, \lambda_{10}, \lambda_{20}, ..., \lambda_{m0}) \le L(x_{10}, x_{20}, ..., x_{n0}, \lambda_{10}, \lambda_{20}, ..., \lambda_{m0}) \le \\ \le L(x_{10}, x_{20}, ..., x_{n0}, \lambda_1, \lambda_2, ..., \lambda_m)$$

для всех $x_j \ge 0 \left(j = \overline{1,n}\right)$ и $\lambda_i \ge 0 \left(i = \overline{1,m}\right)$.

ТЕОРЕМА 5.1. Любой локальный минимум (максимум) задачи выпуклого программирования является глобальным минимумом (максимумом).

ТЕОРЕМА 5.2. (Теорема Куна-Таккера). Для задачи выпуклого программирования (5.5)–(5.6), множество допустимых решений которой обладает свойством регулярности, точка $\boldsymbol{x}_0=(x_{10}, x_{20}, ..., x_{n0})$ является оптимальным решением тогда и только тогда, когда существует такой вектор $\boldsymbol{\lambda}_0 = (\lambda_{10}, \lambda_{20}, ..., \lambda_{m0})$ $(\lambda_{i0} \geq 0, i = \overline{1,m})$, что $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0)$ – седловая точка функции Лагранжа.

Если предположить, что целевая функция f и функции g_i в (5.5)–(5.6) непрерывно дифференцируемы, то теорема Куна-Таккера может быть дополнена аналитическими выражениями, определяющими необходимые и достаточные условия того, чтобы точка $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0)$ была седловой точкой функции Лагранжа, то есть являлась решением задачи выпуклого программирования. Эти выражения имеют следующий вид:

$$\begin{cases} \dfrac{\partial L_0}{\partial x_j} \leq 0 \left(j = \overline{1,n} \right); \\ x_{j0} \dfrac{\partial L_0}{\partial x_j} = 0 \ ; \ x_{j0} \geq 0; \\ \dfrac{\partial L_0}{\partial \lambda_i} \geq 0 \left(i = \overline{1,m} \right); \\ \lambda_{i0} \dfrac{\partial L_0}{\partial \lambda_i} = 0 \ ; \ \lambda_{i0} \geq 0; \end{cases} \qquad (5.9)$$

где $\dfrac{\partial L_0}{\partial x_j}$ и $\dfrac{\partial L_0}{\partial \lambda_i}$ – значения соответствующих частных производных функций Лагранжа, вычисленных в седловой точке. Всем отмеченным выше требованиям, позволяющим записать необходимые и достаточные условия для седловой точки $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0)$ функции Лагранжа в виде выражений (5.9), удовлетворяет задача квадратичного программирования, которая будет сформулирована ниже.

ОПРЕДЕЛЕНИЕ 5.8. Квадратичной формой относительно переменных $x_1, x_2, ..., x_n$ называется скалярная функция от этих переменных, имеющая вид:

$$f = c_{11}x_1x_1 + c_{12}x_1x_2 + c_{13}x_1x_3 + ... + c_{21}x_2x_1 + c_{22}x_2x_2 + ... =$$

$$= \sum_{k=1}^{n} \sum_{j=1}^{n} c_{kj} x_k x_j .$$

(5.10)

ОПРЕДЕЛЕНИЕ 5.9. Квадратичная форма f называется положительно (отрицательно)–определенной, если $f(\boldsymbol{x}) > 0 \quad (f(\boldsymbol{x}) < 0)$ для всех значений переменных $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ кроме $\boldsymbol{x} = 0$. Если же имеют место нестрогие равенства, то квадратичная форма f называется положительно (отрицательно)-полуопределенной.

ТЕОРЕМА 5.3. Квадратичная форма является выпуклой функцией, если она положительно–полуопределенная, и вогнутой функцией, если она отрицательно – полуопределенная.

ОПРЕДЕЛЕНИЕ 5.10. Задача, состоящая в определении минимального (максимального) значения функции

$$\sum_{k=1}^{n} \sum_{j=1}^{n} c_{kj} x_k x_j + \sum_{j=1}^{n} d_j x_j \qquad (5.11)$$

при условиях

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \left(i = \overline{1,m} \right), x_j \ge 0, \qquad (5.12)$$

где $\sum_{k=1}^{n} \sum_{j=1}^{n} c_{kj} x_k x_j$ - положительно (отрицательно) - полуопределенная квадратичная форма, называется задачей квадратичного программирования.

Для задачи (5.11), (5.12) функция Лагранжа будет иметь вид

$$L = \sum_{k=1}^{n} \sum_{j=1}^{n} c_{kj} x_k x_j + \sum_{j=1}^{n} d_j x_j + \lambda_i \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right).$$

Если функция имеет седловую точку $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0)$, то в этой точке выполняются соотношения (5.9). Вводя дополнительные переменные $v_j (j = \overline{1,n})$ и $w_i (i = \overline{1,m})$, перепишем выражения (5.9) в виде

$$\begin{cases} \dfrac{\partial L_0}{\partial x_j} + v_j = 0 \ \left(j = \overline{1,n} \right); \\ \dfrac{\partial L_0}{\partial \lambda_i} - w_i = 0 \ \left(i = \overline{1,m} \right); \\ x_{j0} v_j = 0; \\ \lambda_{i0} w_i = 0; x_{j0} \geq 0, v_j \geq 0, \lambda_{j0} \geq 0, w_j \geq 0. \end{cases} \quad (5.13)$$

5.3.2 Метод неопределенных множителей Лагранжа для решения задач квадратичного программирования

Рассмотрим задачу (5.11), (5.12) из пункта 5.3.1. Чтобы найти решение задачи квадратичного программирования (5.11), (5.12) нужно определить неотрицательное решение систем линейных уравнений (5.13). Это решение можно найти с помощью метода искусственного базиса.

Процесс нахождения решения включает следующие этапы:

1) составляют функцию Лагранжа;

2) записывают в виде системы (5.13) необходимые и достаточные условия существования седловой точки для функции Лагранжа;

3) используя метод искусственного базиса, либо устанавливают отсутствие седловой точки для функции Лагранжа, либо находят ее координаты;

4) записывают оптимальное решение исходной задачи и находят значение целевой функции.

Пример 5.3.

Найти максимальное значение функции

$$f = 2x_1 + 4x_2 - x_1^2 - 2x_2^2 \quad (5.14)$$

при условиях

$$\begin{cases} x_1 + 2x_2 \leq 8 \\ 2x_1 - x_2 \leq 12 \\ x_1 \geq 0, x_2 \geq 0. \end{cases} \quad (5.15)$$

Решение. Функция f является вогнутой, так как представляет собой сумму линейной функции $f_1 = 2x_1 + 4x_2$ (которую можно рассматривать как вогнутую) и квадратичной формы $f_2 = -x_1^2 - 2x_2^2$, которая является отрица-

тельно определенной. Система ограничений включает только линейные неравенства. Следовательно, можно воспользоваться теоремой Куна-Таккера. Составим функцию Лагранжа:

$$L = 2x_1 + 4x_2 - x_1^2 - 2x_2^2 + \lambda_1(8 - 2x_2 - x_1) + \lambda_2(12 + x_2 - 2x_1) \quad (5.16)$$

и запишем необходимые и достаточные условия существования седловой точки построенной функции:

$$\begin{cases} \dfrac{\partial L}{\partial x_1} = 2 - 2x_1 - \lambda_1 - 2\lambda_2 \le 0, \\ \dfrac{\partial L}{\partial x_2} = 4 - 4x_2 - 2\lambda_1 + \lambda_2 \le 0, \\ \dfrac{\partial L}{\partial \lambda_1} = 8 - x_1 - 2x_2 \ge 0, \\ \dfrac{\partial L}{\partial \lambda_2} = 12 - 2x_1 + x_2 \ge 0; \end{cases} \quad (5.17)$$

$$\begin{cases} x_1 \dfrac{\partial L}{\partial x_1} = x_1(2 - 2x_1 - \lambda_1 - 2\lambda_2) = 0, \\ x_2 \dfrac{\partial L}{\partial x_2} = x_2(4 - 4x_2 - 2\lambda_1 + \lambda_2) = 0, \\ \lambda_1 \dfrac{\partial L}{\partial \lambda_1} = \lambda_1(8 - x_1 - 2x_2) = 0, \\ \lambda_2 \dfrac{\partial L}{\partial \lambda_2} = \lambda_2(12 - 2x_1 + x_2) = 0, \\ x_1, x_2, \lambda_1, \lambda_2 \ge 0. \end{cases} \quad (5.18)$$

Систему линейных неравенств (5.17) перепишем в виде:

$$\begin{cases} 2x_1 + \lambda_1 + 2\lambda_2 \ge 2 \\ 4x_2 + 2\lambda_1 - \lambda_2 \ge 4 \\ x_1 + 2x_2 \le 8 \\ 2x_1 - x_2 \le 12. \end{cases} \quad (5.19)$$

Вводя теперь дополнительные неотрицательные переменные v_1, v_2, w_1 и w_2, обращающие неравенства (5.17) в равенства, получим

$$\begin{cases} 2x_1 + \lambda_1 + 2\lambda_2 - v_1 = 2 \\ 4x_2 + 2\lambda_1 - \lambda_2 - v_2 = 4 \\ x_1 + 2x_2 + w_1 = 8 \\ 2x_1 - x_2 + w_2 = 12 \end{cases} \tag{5.20}$$

$$x_1, x_2, \lambda_1, \lambda_2, v_1, v_2, w_1, w_2 \geq 0.$$

Учитывая равенства (5.20), можно записать:

$$v_1 x_1 = 0,\ v_2 x_2 = 0,\ w_1 \lambda_1 = 0,\ w_2 \lambda_2 = 0. \tag{5.21}$$

Если теперь найти базисное решение системы линейных уравнений (5.20) с учетом (5.21), то будет получена седловая точка функции Лагранжа для исходной задачи, то есть определено оптимальное решение.

Для нахождения базисного решения системы линейных уравнений (5.20) воспользуемся методом искусственного базиса. В первое и второе уравнения (5.20) добавим неотрицательные переменные Z_1 и Z_2 и рассмотрим задачу линейного программирования, состоящую в определении максимального значения функции

$$f = -MZ_1 - MZ_2 \tag{5.22}$$

при условиях

$$\begin{cases} 2x_1 + \lambda_1 + 2\lambda_2 - v_1 + z_1 = 2 \\ 4x_2 + 2\lambda_1 - \lambda_2 - v_2 + z_2 = 4 \\ x_1 + 2x_2 + w_1 = 8 \\ 2x_1 - x_2 + w_2 = 12 \end{cases} \tag{5.23}$$

$$x_1, x_2, \lambda_1, \lambda_2, v_1, v_2, w_1, w_2, z_1, z_2 \geq 0. \tag{5.24}$$

В результате решения задачи линейного программирования (5.22)–(5.24) находим допустимое базисное решение системы линейных уравнений (5.23) (таблица 5.1):

Отсюда

$x_{10} = 1,\quad x_{20} = 1,\quad w_1 = 5,\quad w_2 = 11,\quad \lambda_{10} = \lambda_{20} = v_1 = v_2 = 0.$

Так как $x_{10} v_1 = 0,\quad x_{20} v_2 = 0,\quad \lambda_{10} w_1 = 0,\quad \lambda_{20} w_2 = 0,$ то $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0)=(1,1,0,0)$ является седловой точкой функции Лагранжа для исходной задачи. Следовательно, $\boldsymbol{x}^*=(1,1)$ – оптимальный план исходной задачи и $f_{max}=3$.

Таблица 5.1 – Результаты вычислений методом Лагранжа

	Базис	$C_б$	C_0	0 P_{x_1}	0 P_{x_2}	0 $P_{\lambda 1}$	0 $P_{\lambda 2}$	0 P_{v_1}	0 P_{v_2}	0 P_{w_1}	0 P_{w_2}	-M P_{Z_1}	-M P_{Z_2}
1	P_{Z_1}	-M	2	2	0	1	2	-1	0	0	0	1	0
2	P_{Z_2}	-M	4	0	4	2	-1	0	-1	0	0	0	1
3	P_{w_1}	0	8	1	2	0	0	0	0	1	0	0	0
4	P_{w_2}	0	12	2	-1	0	0	0	0	0	1	0	0
5			0	0	0	0	0	0	0	0	0	0	0
6			-6	-2	-4	-3	-1	1	1	0	0	0	0
1	P_{Z_1}	-M	2	2	0	1	2	-1	0	0	0	1	0
2	P_{x_2}	0	1	0	1	1/2	-1/4	0	-1/4	0	0	0	1/4
3	P_{w_1}	0	6	1	0	-1	1/2	0	1/2	1	0	0	-1/2
4	P_{w_2}	0	13	2	0	1/2	-1/4	0	-1/4	0	1	0	-1/4
5		0	0	0	0	0	0	0	0	0	0	0	0
6			-2	-2	0	-1	-2	1	0	0	0	0	1
1	P_{x_1}	0	1	1	0	1/2	1	-1/2	0	0	0	1/2	0
2	P_{x_2}	0	1	0	1					0	0		
3	P_{w_1}	0	5	0	0					1	0		
4	P_{w_2}	0	11	0	0					0	1		
5			0	0	0					0	0		

5.4 Градиентные методы решения задач нелинейного программирования

Градиентные методы позволяют находить приближенное решение любой задачи нелинейного программирования. Однако в общем случае это будет точка локального экстремума. Поэтому более целесообразно использовать градиентные методы для нахождения решения задач выпуклого программирования, в которых локальный экстремум является одновременно и глобальным.

Процесс нахождения решения задачи с помощью градиентных методов состоит в том, что начиная с некоторой точки $\boldsymbol{x}_0$ осуществляется последовательный переход

к некоторым другим точкам до тех пор, пока градиент целевой функции $f(x_1, x_2, ..., x_n)$ в очередной точке $\boldsymbol{x}_{k+1}$ не станет равным нулю, или не будет выполнено условие $|f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)| < \varepsilon$, где ε характеризует точность полученного решения.

Наиболее распространенными из градиентных методов являются методы приведенного градиента Вулфа, штрафных функций и Эрроу–Гурвица.

5.4.1 Метод приведенного градиента Вулфа

Пусть требуется минимизировать целевую функцию

$$f(\boldsymbol{x}) \tag{5.25}$$

при условиях

$$\begin{cases} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \boldsymbol{x} \geq 0. \end{cases} \tag{5.26}$$

где матрица $\boldsymbol{A}$ – матрица порядка $m \times n$ и ранга m, $\boldsymbol{b}$ – вектор из E_m, функция f непрерывно дифференцируема в E_n.

Заметим, что ограничения (5.26) линейные. Предположим, что $\boldsymbol{A}$ невырожденная матрица, а $\boldsymbol{x}$ - допустимая точка. Матрицу A можно представить в виде $[\boldsymbol{B},\boldsymbol{D}]$, а вектор $\boldsymbol{x}$ – в виде $[\boldsymbol{x}_B, \boldsymbol{x}_D]$, где $\boldsymbol{B}$ – неособенная матрица порядка $m \times m$. Вектор $\boldsymbol{x}_B$ называется базисным, и его компоненты строго положительны. Компоненты внебазисного вектора $\boldsymbol{x}_D$ могут быть либо положительными, либо нулевыми.

Пусть $\nabla f(\boldsymbol{x}) = [\nabla_B f(\boldsymbol{x}), \nabla_D f(\boldsymbol{x})]$, где $\nabla_B f(\boldsymbol{x})$ – градиент функции f по переменным базисного вектора $\boldsymbol{x}_B$, а $\nabla_D f(x)$ – градиент f по внебазисным переменным $\boldsymbol{x}_D$. Направление $\boldsymbol{d}$ является направлением спуска и возможным для функции f в точке $\boldsymbol{x}$, если $\nabla f(\boldsymbol{x})\boldsymbol{d} < 0$, $\boldsymbol{A}\boldsymbol{d} = 0$ и $d_j \geq 0$, если $x_j = 0$. Определим вектор $\boldsymbol{d}$, обладающий этими свойствами. Представим вектор $\boldsymbol{d}$ в виде $[\boldsymbol{d}_B, \boldsymbol{d}_D]$. Равенство $\boldsymbol{A}\boldsymbol{d} = \boldsymbol{B}\boldsymbol{d}_B + \boldsymbol{D}\boldsymbol{d}_D = 0$ выполняется, если для любого $\boldsymbol{d}_D$ положить $\boldsymbol{d}_B = -\boldsymbol{B}^{-1}\boldsymbol{D}\boldsymbol{d}_D$.

Пусть $\boldsymbol{r}=(\boldsymbol{r}_B,\boldsymbol{r}_D)=\nabla f(\boldsymbol{x})-\nabla_B f(\boldsymbol{x})\boldsymbol{B}^{-1}\boldsymbol{A}$ – приведенный градиент. Исследуем $\nabla f(\boldsymbol{x})\boldsymbol{d}$.

$$\nabla f(\boldsymbol{x})\boldsymbol{d}=\nabla_B f(\boldsymbol{x})\boldsymbol{d}_B+\nabla_D f(\boldsymbol{x})\boldsymbol{d}_D=\left[\nabla_D f(\boldsymbol{x})-\nabla_B f(\boldsymbol{x})\boldsymbol{B}^{-1}\boldsymbol{N}\right]\boldsymbol{d}_D=$$
$$=\boldsymbol{r}_D\boldsymbol{d}_D.$$

Мы должны выбрать вектор $\boldsymbol{d}_D$ так, чтобы $\boldsymbol{r}_D\boldsymbol{d}_D<0$ и $d_j\geq 0$, если $x_j=0$. Для этого принимается следующее правило. Для каждой внебазисной компоненты j положим $d_j=-r_j$, если $r_j\leq 0$, и положим $d_j=-r_j x_j$, если $r_j>0$. Из этого следует выполнение неравенства $d_j\geq 0$, если $x_j=0$. Кроме того, $\nabla f(\boldsymbol{x})\boldsymbol{d}\leq 0$ и строгое неравенство имеет место, если $\boldsymbol{d}_D\neq 0$.

Здесь вектор $\boldsymbol{d}=0$ в том и только том случае, если $\boldsymbol{x}$ седловая точка Куна–Таккера.

В целях сокращения записей символ, означающий транспонирование опущен.

Алгоритм метода приведенного градиента.

Пусть имеем задачу (5.26). Предположим, что любые m столбцов матрицы $\boldsymbol{A}$ линейно независимы и что каждая экстремальная точка допустимой области (5.26) имеет m строго положительных компонент. Указанный алгоритм сходится к точке Куна–Таккера при условии, что в качестве базисных переменных выбраны m наибольших положительных переменных.

Начальный этап.

Выбрать точку $\boldsymbol{x}_1$, удовлетворяющую условиям $\boldsymbol{A}\boldsymbol{x}_1=\boldsymbol{b},\boldsymbol{x}_1\geq 0$. Положить k=1 и перейти к основному этапу.

Основной этап.

Шаг 1. Положить $\boldsymbol{d}_k^T=(\boldsymbol{d}_B,\boldsymbol{d}_D)^T$, где $\boldsymbol{d}_D$ и $\boldsymbol{d}_B$ получаются из формул (5.30) и (5.31) соответственно. Если $\boldsymbol{d}_k=0$, то остановиться; $\boldsymbol{x}_k$– оптимальная точка (точка Куна – Таккера). В противном случае перейти к шагу 2.

I_k – множество индексов m наибольших компонент вектора $\boldsymbol{x}_k$, (5.27)

$$\boldsymbol{B}=\{a_j : j\in I_k\},\ \boldsymbol{D}=\{a_j : j\notin I_k\}, \tag{5.28}$$

$$\boldsymbol{r}^T=\nabla f(\boldsymbol{x}_k)^T-\nabla_B f(\boldsymbol{x}_k)^T\boldsymbol{B}^{-1}\boldsymbol{A}, \tag{5.29}$$

$$d_j=\begin{cases}-r_j, \text{если } j\notin I_k \quad и \quad r_j\le 0,\\ -x_j r_j, \text{если } j\notin I_k \quad и \quad r_j>0,\end{cases} \tag{5.30}$$

$$\boldsymbol{d}_B=-\boldsymbol{B}^{-1}\boldsymbol{D}\boldsymbol{d}_D. \tag{5.31}$$

Шаг 2. Решить следующую задачу одномерной минимизации:
минимизировать

$$f(\boldsymbol{x}_k+\lambda\boldsymbol{d}_k)$$

при условии

$$0\le\lambda\le\lambda_{\max},$$

где
$$\lambda_{\max}=\begin{cases}\min\limits_{1\le j\le n}\left\{-\dfrac{x_{jk}}{d_{jk}} : d_{jk}<0\right\}, \text{если} \quad \boldsymbol{d}_k<0,\\ \infty, \qquad\qquad \text{если} \quad \boldsymbol{d}_k\ge 0.\end{cases} \tag{5.32}$$

Здесь x_{jk}, d_{jk} – j-е компоненты векторов $\boldsymbol{x}_k$ и $\boldsymbol{d}_k$.

Положим λ_k равным оптимальному и $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\lambda_k\boldsymbol{d}_k$. Заменить k на $k+1$ и перейти к шагу 1.

Пример 5.4.

Рассмотрим задачу:

минимизировать $\quad 2x_1^2+2x_2^2-2x_1x_2-4x_1-6x_2$

при условиях

$$x_1+x_2+x_3=2$$
$$x_1+5x_2+x_4=5$$
$$x_1,x_2,x_3,x_4\ge 0.$$

В качестве начальной точки выберем $\boldsymbol{x}_1=(0,0,2,5)^T$.

Градиент функции $\nabla f(\boldsymbol{x})=(4x_1-2x_2-4,\ 4x_2-2x_1-6,0,0)^T$.

Информацию по каждой итерации будем представлять

в виде таблицы, подобной симплекс-таблице.

Итерация 1.

Поиск направления.

В точке $\boldsymbol{x}_1 = (0, 0, 2, 5)^T$ имеем $\nabla f(\boldsymbol{x}_1) = (-4, -6, 0, 0)^T$. В соответствии с (5.27) множество $I_1=\{3,4\}$, так что $\boldsymbol{B}=[a_3, a_4]$ и $\boldsymbol{D}=[a_1, a_2]$ (согласно (5.28)). Согласно (5.29) приведенный градиент

$$\boldsymbol{r}^T = (-4,-6, 0, 0) - (0, 0)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 5 & 0 & 1 \end{pmatrix} =$$

$$= (-4,-6, 0, 0) - (0, 0)\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 5 & 0 & 1 \end{pmatrix} = (-4,-6, 0, 0).$$

Результаты вычислений будем помещать в таблицу 5.2.

В соответствии с (5.30) имеем $\boldsymbol{d}_D=(d_1,d_2)^T=(4,6)^T$. По формуле (5.31) $\boldsymbol{d}_B=(d_3,d_4)^T=-\boldsymbol{B}^{-1}\boldsymbol{D}\boldsymbol{d}_D$. Получим

$$\boldsymbol{B} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \boldsymbol{B}^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \boldsymbol{D}=\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Тогда $\boldsymbol{d}_B = -\begin{pmatrix} 1 & 1 \\ 1 & 5 \end{pmatrix}\begin{pmatrix} 4 \\ 6 \end{pmatrix} = (-10,-34)^T$. Вектор направления, таким образом, равен $\boldsymbol{d}_1 = (4, 6, -10, -34)^T$.

Линейный поиск.

При начальной точке $(0, 0, 2, 5)^{\mathrm{T}}$ минимизируем целевую функцию по направлению $\boldsymbol{d}_1 = (4, 6, -10, -34)^T$. Максимальное значение λ, для которого точка $\boldsymbol{x}_1 + \lambda\boldsymbol{d}_1$ допустима, вычисляется в соответствии с (5.32) и равно

$$\lambda_{\max} = \min\left\{\frac{2}{10}, \frac{5}{34}\right\} = \frac{5}{34}.$$

Значение целевой функции в точке $\boldsymbol{x}_1 + \lambda\boldsymbol{d}_1$

$$f(\boldsymbol{x}_1 + \lambda\boldsymbol{d}_1) = 56\lambda^2 - 52\lambda,$$

так что задача линейного поиска имеет вид:

минимизировать

$$56\lambda^2 - 52\lambda$$

при условии

$$0 \le \lambda \le \frac{5}{34}.$$

Отсюда

$$\lambda_1 = \frac{5}{34} \text{ и } \boldsymbol{x}_2 = \boldsymbol{x}_1 + \lambda_1 \boldsymbol{d}_1 = \left(\frac{10}{17}, \frac{15}{17}, \frac{9}{17}, 0 \right)^T.$$

Итерация 2.

Поиск направления. В точке $\boldsymbol{x}_2 = \left(\frac{10}{17}, \frac{15}{17}, \frac{9}{17}, 0 \right)^T$ в соответствии с (5.27) имеем

$$I_2 = \{1,2\},\ \boldsymbol{B} = [a_1, a_2],\ \boldsymbol{D} = [a_3, a_4] \text{ и}$$

$$\nabla f(\boldsymbol{x}_2) = \left(-\frac{58}{17}, -\frac{62}{17}, 0, 0 \right)^T.$$

В соответствии с (5.29) имеем

$$\boldsymbol{r}^T = \left(-\frac{58}{17}, -\frac{62}{17}, 0, 0 \right) - \left(-\frac{58}{17}, -\frac{62}{17} \right) \begin{pmatrix} 1 & 1 \\ 1 & 5 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 5 & 0 & 1 \end{pmatrix} =$$

$$= \left(-\frac{58}{17}, -\frac{62}{17}, 0, 0 \right) - \left(-\frac{58}{17}, -\frac{62}{17} \right) \begin{pmatrix} 1 & 0 & \frac{5}{4} & -\frac{1}{4} \\ 0 & 1 & -\frac{1}{4} & \frac{1}{4} \end{pmatrix} = \left(0,0, \frac{57}{17}, \frac{4}{17} \right).$$

Тогда, согласно (5.30) имеем, что

$\boldsymbol{d}_3 = -\left(\frac{9}{17} \right)\left(\frac{57}{17} \right) = -\frac{513}{289}$ и $\boldsymbol{d}_4 = 0$, так что $\boldsymbol{d}_D = \left(-\frac{513}{289}, 0 \right)^T$.

Вектор $\boldsymbol{d}_B$ равен

$$\boldsymbol{d}_B = (d_1, d_2)^T = \begin{pmatrix} \frac{5}{4} & -\frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} -\frac{513}{289} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{2565}{1156} \\ -\frac{513}{1156} \end{pmatrix}.$$

Таким образом, вектор направления равен

$$\boldsymbol{d}_2 = \left(\frac{2565}{1156}, -\frac{513}{1156}, -\frac{513}{289}, 0 \right)^T .$$

Линейный поиск.

Начиная процедуру из точки $\boldsymbol{x}_2 = \left(\frac{10}{17}, \frac{15}{17}, \frac{9}{17}, 0 \right)^T$, минимизируем целевую функцию по направлению $\boldsymbol{d}_2 = \left(\frac{2565}{1156}, -\frac{513}{1156}, -\frac{513}{289}, 0 \right)^T$. Максимальное значение λ, для которого точка $\boldsymbol{x}_2 + \lambda \boldsymbol{d}_2$ допустима, равно

$$\lambda_{\max} = \min \left\{ \frac{-\frac{15}{17}}{-\frac{513}{1156}}, \frac{-\frac{9}{17}}{-\frac{513}{289}} \right\} = \frac{17}{57}.$$

Значение целевой функции в точке $\boldsymbol{x}_2 + \lambda \boldsymbol{d}_2$ $f(\boldsymbol{x}_2 + \lambda \boldsymbol{d}_2) = 12{,}21\lambda^2 - 5{,}95\lambda - 6{,}436$, так что λ_2 находится из решения задачи.

Минимизировать

$$12{,}21\lambda^2 - 5{,}95\lambda - 6{,}436$$

при условии

$$0 \le \lambda \le \frac{17}{57}.$$

Отсюда $\lambda_2 = \frac{68}{279}$, $\boldsymbol{x}_3 = \boldsymbol{x}_2 + \lambda_2 \boldsymbol{d}_2 = \left(\frac{35}{31}, \frac{24}{31}, \frac{3}{31}, 0 \right)^T$.

Итерация 3.

Поиск направления.

Теперь $I_3 = \{1,2\}$, то есть $\boldsymbol{B} = [a_1, a_2]$ и $\boldsymbol{D} = [a_3, a_4]$. Имеем

$$\nabla f(\boldsymbol{x}_3) = \left(-\frac{32}{31}, -\frac{160}{31}, 0, 0 \right)^T .$$

Вектор

$$\boldsymbol{r}^T = \left(-\frac{32}{31}, -\frac{160}{31}, 0, 0 \right) - \left(-\frac{32}{31}, -\frac{160}{31} \right) \begin{pmatrix} 1 & 1 \\ 1 & 5 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 5 & 0 & 1 \end{pmatrix} = (0, 0, 0, 1)$$

Отсюда $\boldsymbol{d}_D = (d_3, d_4)^T = (0,0)^T$, $\boldsymbol{d}_B = (d_1, d_2)^T = (0,0)^{\mathbf{T}}$. Следовательно, $\boldsymbol{d} = 0$ и решение $\boldsymbol{x}_3$ оптимально.

Таблица 5.2 – Результаты вычислений методом градиента Вулфа

	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$f(\boldsymbol{x}_k)$
Решение $\boldsymbol{x}_1$	0	0	2	5	0,0
$\nabla f(\boldsymbol{x}_k)$	-4	-6	0	0	
$\nabla_B f(\boldsymbol{x}_1) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $\begin{matrix} \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \end{matrix}$	$\begin{matrix} 1 \\ 1 \end{matrix}$	$\begin{matrix} 1 \\ 5 \end{matrix}$	$\begin{matrix} 1 \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 1 \end{matrix}$	
$\boldsymbol{r}_1$	-4	-6	0	0	
Решение $\boldsymbol{x}_2$	$\frac{10}{17}$	$\frac{15}{17}$	$\frac{9}{17}$	0	-6,436
$\nabla f(\boldsymbol{x}_2)$	$-\frac{58}{17}$	$\frac{62}{17}$	0	0	
$\nabla_B f(\boldsymbol{x}_2) = \begin{pmatrix} -\frac{58}{17} \\ -\frac{62}{17} \end{pmatrix}$ $\begin{matrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{matrix}$	$\begin{matrix} 1 \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 1 \end{matrix}$	$\begin{matrix} \frac{5}{4} \\ -\frac{1}{4} \end{matrix}$	$\begin{matrix} -\frac{1}{4} \\ \frac{1}{4} \end{matrix}$	
$\boldsymbol{r}_2$	0	0	$\frac{57}{17}$	$\frac{4}{17}$	
Решение $\boldsymbol{x}_3$	$\frac{35}{31}$	$\frac{24}{31}$	$\frac{3}{31}$	0	-7,16
$\nabla f(\boldsymbol{x}_3)$	$-\frac{32}{31}$	$-\frac{160}{31}$	0	0	
$\nabla_B f(\boldsymbol{x}_3) = \begin{pmatrix} -\frac{32}{31} \\ -\frac{160}{31} \end{pmatrix}$ $\begin{matrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{matrix}$	$\begin{matrix} 1 \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 1 \end{matrix}$	$\begin{matrix} \frac{5}{4} \\ -\frac{1}{4} \end{matrix}$	$\begin{matrix} -\frac{1}{4} \\ \frac{1}{4} \end{matrix}$	
$\boldsymbol{r}_3$	0	0	0	1	

5.4.2 Метод штрафных функций

Для определенности рассмотрим задачу нахождения максимального значения вогнутой функции $f(x_1, x_2, ..., x_n)$ при условиях $g_i(x_1, x_2, ..., x_n) \le b_i (i = \overline{i,m}), \quad x_j \ge 0 \quad (j = \overline{1,n})$, где g_i – выпуклые функции. Заметим, что здесь функции g_i могут быть как линейными, так и нелинейными. Однако в первом случае удобнее использовать метод приведенного градиента Вулфа, что существенно сокращает вычисления.

Вместо непосредственного решения вышеприведенной задачи, находят максимальное значение функции $F(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n) + H(x_1, x_2, ..., x_n)$, являющейся суммой целевой функции задачи, и некоторой функции H, определяемой системой ограничений и называемой **штрафной функцией**. Штрафную функцию записываем в виде:

$$H(x_1, x_2, ..., x_n) = \sum_{i=1}^{m} \alpha_i (x_1, x_2, ..., x_n)\, g_i (x_1, x_2, ..., x_n),$$

где $$\alpha_i (x_1, x_2, ..., x_n) = \begin{cases} 0, если \quad b_i - g_i (x_1, x_2, ..., x_n) \ge 0 \\ \alpha_i, если \quad b_i - g_i (x_1, x_2, ..., x_n) < 0 \end{cases}, \quad (5.33)$$

а $\alpha_i > 0$ – некоторые постоянные числа, называемые весомыми коэффициентами.

Используя штрафную функцию, последовательно переходят от одной точки к другой до тех пор, пока не получат приемлемое решение. При этом координаты последующей точки находят по формуле

$$x_j^{(k+1)} = \max \left\{ 0,\ x_j^{(k)} + \lambda \left[\frac{\partial f(x^{(k)})}{\partial x_j} + \sum_{i=1}^{m} \alpha_i \frac{\partial g(x^{(k)})}{\partial x_j} \right] \right\}. \quad (5.34)$$

Из соотношения (5.34) следует, что если предыдущая точка находится в области допустимых решений исходной задачи, то второе слагаемое в квадратных скобках

равно нулю и переход к последующей точке определяется только градиентом целевой функции.

Если же указанная точка не принадлежит области допустимых решений, то за счет данного слагаемого на последующих итерациях достигается возвращение в область допустимых решений. При этом значении α_i должны быть маленькими.

Алгоритм метода штрафных функций.

Процесс нахождения решения задачи выпуклого программирования включает следующие этапы:

1. Определяют исходное допустимое решение.

2. Задают точность вычислений (штрафной параметр λ).

3. Находят по всем переменным частные производные от целевой функции и функций, определяющих область допустимых решений задачи.

4. По формулам (5.34) находят координаты точки, определяющей возможное новое решение задачи.

5. Проверяют, удовлетворяют ли координаты найденной точки системе ограничений задачи. Если нет, то переходят к следующему этапу. Если координаты найденной точки удовлетворяют системе ограничений, то исследуют необходимость перехода к последующему допустимому решению. В случае такой необходимости переходят к этапу 2, в противном случае найдено решение задачи с заданной точностью.

6. Устанавливают значение весовых коэффициентов и переходят к этапу 4.

Пример 5.5.

Найти максимальное значение функции $f = -x_1^2 - x_2^2$ при условиях

$$(x_1 - 7)^2 + (x_2 - 7)^2 \le 18,$$

$$x_1, x_2 \ge 0.$$

Целевая функция данной задачи представляет собой отрицательно–определенную квадратичную форму, а ограничения – выпуклую область. Следовательно, имеем задачу выпуклого программирования. Построим область

допустимых решений задачи и линии уровня (рисунок 5.5). Точка касания одной из этих окружностей с областью допустимых решений и является искомой точкой максимума целевой функции.

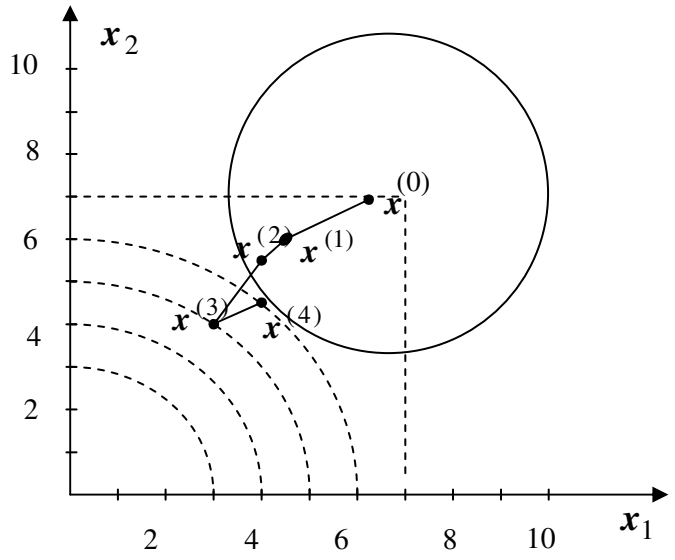


Рисунок 5.5 – Область допустимых значений и линии уровня

В качестве начальной точки возьмем $\boldsymbol{x}^{(0)}=(6,7)$ и положим $\lambda=0,1$. Определим частные производные:

$$\frac{\partial f}{\partial x_1}=-2x_1; \quad \frac{\partial f}{\partial x_2}=-2x_2; \quad \frac{\partial g}{\partial x_1}=-2x_1+14; \quad \frac{\partial g}{\partial x_2}=-2x_2+14.$$

Далее, используя формулу (5.34), построим последовательность точек для определения приемлемого решения.

Итерация 1.

Так как точка $\boldsymbol{x}^{(0)}=(6,7)$ принадлежит области допустимых решений, то

$$x_1^{(1)}=\max\left\{0; x_1^{(0)}+\lambda\frac{\partial f(\boldsymbol{x}^{(0)})}{\partial x_1}\right\}=\max\{0;6+0,1\cdot(-2)\cdot 6\}=4,8,$$

$$x_2^{(1)}=\max\left\{0; x_2^{(0)}+\lambda\frac{\partial f(\boldsymbol{x}^{(0)})}{\partial x_2}\right\}=\max\{0;7+0,1(-2)7\}=5,6.$$

Найдем $g(\boldsymbol{x}^{(1)}) = 18 - 4{,}84 - 1{,}96 = 11{,}2$. Так как $g(\boldsymbol{x}^{(1)}) > 0$, то точка $\boldsymbol{x}^{(1)} = (4{,}8\,;5{,}6)$ принадлежит допустимой области. В этой точке значение целевой функции $f(\boldsymbol{x}^{(1)}) = -54{,}4$.

Итерация 2.

$$x_1^{(2)} = \max\{0\,;4{,}8 + 0{,}1(-2)4{,}8\} = 3{,}84$$

$$x_2^{(2)} = \max\{0\,;5{,}6 + 0{,}1(-2)5{,}6\} = 4{,}48$$

$$g(\boldsymbol{x}^{(2)}) = 18 - 9{,}9856 - 6{,}3504 = 1{,}664 > 0\,;\ f(\boldsymbol{x}^{(2)}) = -34{,}816.$$

Итерация 3.

$$x_1^{(3)} = \max\{0\,;3{,}84 + 0{,}1(-2)3{,}84\} = 3{,}072$$

$$x_2^{(3)} = \max\{0;\,4{,}48 + 0{,}1(-2)4{,}48\} = 3{,}584$$

$$g(\boldsymbol{x}^{(3)}) = 18 - 15{,}429184 - 11{,}669056 \approx -9{,}0981 < 0.$$

Итерация 4.

Точка $\boldsymbol{x}^{(3)}$ не принадлежит допустимой области. Следовательно

$$x_1^{(4)} = \max\left\{0\,; x_1^{(3)} + \lambda\left[\frac{\partial f(\boldsymbol{x}^{(3)})}{\partial x_1} + \alpha\frac{\partial g(\boldsymbol{x}^{(3)})}{\partial x_1}\right]\right\} =$$

$$= \max\{0\,;3{,}072 - 0{,}1[(-2)\cdot 3{,}072 + \alpha\cdot((-2)\cdot 3{,}072 + 14)]\} =$$

$$= \max\{0\,;2{,}4576 + \alpha\cdot 0{,}7856\}.$$

Аналогично получим:

$$x_2^{(4)} = \max\{0;\ 2{,}6672 + \alpha\cdot 0{,}6832\}.$$

Здесь возникает вопрос о выборе весового коэффициента α. Выберем его так, чтобы точка $\boldsymbol{x}^{(4)}$ не слишком далеко удалилась от границы допустимой области и вместе с тем принадлежала этой области. Этим требованиям удовлетворяет $\alpha = 1{,}9$. Тогда $x_1^{(4)} \approx 3{,}910$; $x_2^{(4)} \approx 4{,}165$; $g(\boldsymbol{x}^{(4)}) \approx 0{,}660$; $f(\boldsymbol{x}^{(4)}) = -32{,}950$.

Результаты вычислений приведены в таблице 5.3.

Таблица 5.3 – Результаты вычислений методом штрафных функций

K	$\boldsymbol{x}^{(k)}$	$g(\boldsymbol{x}^{(k)})$	$f(\boldsymbol{x}^{(k)})$
1	(4,8; 5,6)	11,2	-54,4
2	(3,84; 4,48)	1,664	-34,816
3	(3,072; 3,584)	-9,0981	
4	(3,950; 4,165)	0,660	-32,950
5	(3,16; 3,332)	-10,2	
6	(3,987; 4,059)	0,272	-32,372
7	(3,189; 3,247)	-2,715	
8	(3,999; 4,027)	-0,137	-32,185
9	(3,199; 3,219)	-10,744	
10	(4,004; 4,012)	0,096	-32,128
11	(3,203; 3,210)	-10,781	
12	(4,005; 4,008)	0,078	-32,104

Сравнивая значения целевой функции, полученные в 10-й и 12-й итерациях, видим, что они совпадают с точностью до 10^{-1}. Это говорит о близости точки $\boldsymbol{x}^{(12)} = (4{,}005; 4{,}008)$ к точке максимума целевой функции. Исследуем теперь градиенты функций $f(\boldsymbol{x})$ и $g(\boldsymbol{x})$ в точке $\boldsymbol{x}^{(12)}$:

$$\nabla f(\boldsymbol{x}^{(12)}) = (-8{,}01; -8{,}016), \quad \nabla g(\boldsymbol{x}^{(12)}) = (5{,}99; 5{,}984).$$

Вычислим отношения одноименных координат векторов: $-8{,}01/5{,}99 \approx -1{,}337$; $-8{,}016/5{,}984 \approx -1{,}339$. Следовательно, векторы $\nabla f(\boldsymbol{x}^{(12)})$ и $\nabla g(\boldsymbol{x}^{(12)})$ практически коллинеарны. Полученное решение $\boldsymbol{x}^{(12)} = (4{,}005; 4{,}012)$ при необходимости можно уточнить дальнейшими шагами до полной коллинеарности градиентов целевой и ограничительной функций.

В заключении заметим, что на практике используются несколько вариантов штрафных функций, а также то, что решение сложных задач нелинейного программирования градиентными методами связано с большим объемом вычислений и целесообразно только с использованием ЭВМ.

6 Лабораторные работы

6.1 Лабораторная работа №1. Решение задачи линейного программирования

1. Решение задачи геометрическим способом.
2. Решение задачи с помощью симплекс-таблицы.
3. Решение задачи с помощью программы simplecs.

Входные данные:

- N – число переменных;
- M – число ограничений;
- eps – точность;
- ip–признак вида задачи (если на максимум ,то ip=1,если на минимум, то ip=0);
- T[i] – коэффициенты целевой функции;
- S – массив из M–2 чисел ,содержащий правые части системы;
- R – массив из (M–2)* N чисел, содержащий коэффициенты при неизвестных в системе ограничений.

Выходные данные:

- ip–признак окончания решения(ip=1–найдено оптимальное решение; ip=2–задача не имеет решения; ip =3– целевая функция не ограничена);
- nb– массив ,содержащий номера переменных в массиве x;
- x–массив из M чисел ,содержащий оптимальное решение;
- f–оптимальное значение целевой функции.

Пример. Найти максимальное значение целевой функции $f(\mathbf{x}) = 10x_1 - x_2 - 9x_3 - 8x_4$

$$\begin{cases} -2x_1 + x_2 + 3x_3 + x_4 = 2 \\ -5x_1 + 2x_2 + 3x_4 = 5 \\ 7x_1 - 4x_2 + x_3 + 4x_4 \le 1 \\ 3x_1 + 2x_2 + 5x_3 + 6x_4 \le 10 \end{cases}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0, x_4 \ge 0.$$

Текст программы.

```
program  simplecs;
type mas=array[1..100] of real;
     mas1=array[1..100] of integer;
     var
        r,s,t,a,u,x,xk:mas;
        nb:mas1;
       i,l,k,z1,mi,m1,ni,ne,ip,m,n:integer;
        eps,tmin,teta:real;
procedure sol00(r,s,t:mas;var a,x:mas; n,m:integer);
   var k1,k2,k3,j,mj,l,i:integer;
   BEGIN
      l:=m-2;
   for j:=1 to n do
   begin
   mj:=m*j;
   a[mj]:=0;
   for i:=1 to l do
   begin
   k1:=m*(j-1)+i;
   k2:=l*(j-1)+i;
   a[k1]:=r[k2];
   a[mj]:=a[mj]-r[k2];
   end;
   end;
   for i:=1 to n do
   begin
   k3:=m*i-1;
   a[k3]:=t[i];
   end;
   x[m-1]:=0;
   x[m]:=0;
   for i:=1 to l do
   begin
   x[i]:=s[i];
   x[m]:=x[m]-x[i];
end;END;
 procedure sol01(var u:mas;m:integer);
 var i,j,l:integer;
    BEGIN
    for j:=1 to m do
    for i:=1 to m do
    begin
      l:=m*(j-1)+i;
```

```
    u[l]:=0;
    if (i-j)=0 then
    u[l]:=1;
    end;
    END;
procedure sol02(u,a:mas;m,n,j:integer;var del:real);
var i,im,ij:integer;
begin
del:=0;
for i:=1 to m do
begin
im:=i*m;
ij:=m*(j-1)+i;
del:=del+u[im]*a[ij];
end;
EnD;
procedure sol03(var tmin:real;a,u:mas;nb:mas1;m,n:integer;var k:integer);
           var  bul,i,j,m1:integer;
           del:real;
  begin
  tmin:=0;
  m1:=m-2;
  for j:=1 to n do
  begin
 bul:=1;
 i:=1;
 while (bul=1) and (i<=m1) do
 if (j-nb[i])=0  then bul:=2
 else
 i:=i+1;
 if bul<>2 then
 begin
 sol02(u,a,m,n,j,del);
 if (del-tmin)<=0 then
 begin
 tmin:=del;
 k:=j;
 end;
 end;
 end;
 end;
 procedure sol04(u,a:mas;m,n,k:integer;var xk:mas);
 var
    ij,jk,i,j:integer;
  begin
```

```
    for i:=1 to m do
    begin
    xk[i]:=0;
    for j:=1 to m do
    begin
    ij:=m*(j-1)+i;
    jk:=m*(k-1)+j;
    xk[i]:=xk[i]+u[ij]*a[jk];
    end;
    end;
    end;
procedure sol05(x,xk:mas;m:integer;var l:integer;var teta:real;eps:real);
var
  i,m1:integer;
  r:real;
begin
  teta:=10000;
  m1:=m-2;
   for i:=1 to m1 do
   if(xk[i]-eps)>=0 then
   begin
   r:=x[i]/xk[i];
  if (r-teta)<=0 then
  begin
  teta:=r;
  l:=i;
  end;
  end;
  end;
procedure sol06(var x,xk:mas;m,l:integer;var teta:real);
var i:integer;
begin
  for i:=1 to m do begin
  If (i-l)<>0 then
  x[i]:=x[i]-teta*xk[i]
  else
  x[i]:=teta;
  end;
  end;
procedure sol07(var u:mas;m,l:integer;xk:mas);
var
      m1,j,lj,i,ij:integer;
     begin
  m1:=m-2;
  for j:=1 to m1 do
```

```
   begin
   lj:=m*(j-1)+l;
   u[lj]:=u[lj]/xk[l];
   end;
   for i:=1 to m do
   for j:=1 to m1 do
   if (i-l)<>0 then
   begin
   ij:=m*(j-1)+i;
   lj:=m*(j-1)+l;
   u[ij]:=u[ij]-u[lj]*xk[i];
   end;
   end;
 procedure sol08(u,a:mas;m,j:integer;var del:real);
 var mi,ij,i:integer;
 begin
 del:=0;
   for i:=1 to m do
   begin
   mi:=m*i-1;
   ij:=m*(j-1)+i;
   del:=del+u[mi]*a[ij];
   end;
   end ;
procedure sol09(var tmin:real;var k:integer;a,u:mas;nb:mas1;m,n:integer);
var bul,m1,i,j:integer;
    del:real;
 begin
 tmin:=0;
 m1:=m-2;
 for j:= 1 to n do
 begin
 bul:=1;
 i:=1;
 while (bul=1)  and (i<=m1) do
 if (j-nb[i])=0 then
 bul:=2
 else
 i:=i+1;
 if bul<>2 then
 begin
 sol08(u,a,m,j,del);
 if (del-tmin)<=0 then
 begin
 tmin:=del;
```

```
   k:=j;
   end;
   end;
   end;
   end;
   end;
procedure sol10(var tmin:real;u,a:mas;nb:mas1;m,n:integer;var k:integer;
eps:real);
 var bul,m1,i,j:integer;
   del,del1:real;
   begin
       tmin:=0;
       m1:=m-2;
    for j:=1 to n do
      begin
       bul:=1;
       i:=1;
     while (bul=1) and (i<=m1) do
        if (j-nb[i])=0 then
          bul:=2
          else
     i:=i+1;
     if bul<>2 then
     begin
     sol02(u,a,m,n,j,del);
     sol08(u,a,m,j,del1);
     if (abs(del1)-eps)<=0 then
     if (del-tmin)<=0 then
     begin
     tmin:=del;
     k:=j;
     end;
     end;
     end;
     end;
BEGIN
  write(‘ n’);
  read(n);
  writeln(‘m’);readln(m);
  writeln(' (eps)=>');read(eps);
  writeln('ip(ip=1 if MAKS;ip=0 if MIN )=>');
  read(ip);
  for i:=1 to n do
  begin
    writeln('t[',i, ']= ');
    read(t[i]);
```

```
   end;
  for i:=1 to m-2 do
  begin
  writeln('s[',i, ']= ');
  read(s[i]);
  end;
  for i:=1 to (m-2)*n do
  begin
   writeln('r[',i, ']= ' );
  read(r[i]);end;
  sol00(r,s,t,a,x,n,m);
  if (ip-1)=0 then
  for i:=1 to n do
  begin
   mi:=m*i-1;
   a[mi]:=-a[mi];
 end;
   sol01(u,m);
   m1:=m-2;
   for i:=1 to m1 do
   nb[i]:=100011+i;
   ni:=0;
   ne:=1;
3: sol03(tmin,a,u,nb,m,n,k);
2:   if (tmin+eps)>=0 then
   if (ne)=1 then
   if (x[m]+eps)>=0 then
   begin
   ne:=2;
   for i:=1 to m1 do
   if (nb[i]-10000)>0 then
   ne:=3 ;
   if ne=3 then
   begin
   sol10(tmin,u,a,nb,m,n,k,eps);
    goto 2;
    end  else
    begin
    sol09(tmin,k,a,u,nb,m,n);
    goto 2;
    end;end else
    begin
    ip:=2;
    goto 10;
    end
```

```
   else
   if (ip-1)<>0 then
   begin
   x[m-1]:=-x[m-1];
   ip:=1;
   goto 10;
   end
   else
   begin
   ip:=1;
   goto 10;
   end
   else
   begin
   sol04(u,a,m,n,k,xk);
   sol05(x,xk,m,l,teta,eps);
   if (teta+5-10000)<0 then
   begin
   sol06(x,xk,m,l,teta);
   sol07(u,m,l,xk);
   nb[l]:=k;
   ni:=ni+1;
   if ne<>1 then
   if ne=2 then
   begin
   sol09(tmin,k,a,u,nb,m,n);
   goto 2;
   end
   else
   begin
   sol10(tmin,u,a,nb,m,n,k,eps);
   goto 2;
   end
   else
   goto 3 ;end
    else
    begin
    ip:=3;
    goto 10;
    end; end;
10:  writeln('ip=',ip);
 if ip=1 then begin
   for i:=1 to m-2 do
       writeln('x[', nb[i], ']=',x[i]:13);
       for i:=1  to m-2  do
```

```
        writeln('nb[',i,']=',nb[i]);end;
        writeln('f=', x[m-1]:13);
        readln; readln;readln;
END.
```

Входные данные:

```
N=6
M=6
EPS=0.1e–6
ip=1
10 –1 –9  –8  0  0
 2 5 1 10
–2  1  3  1  0  0
–5  2  0  3  0  0
 7  –4  1  4  1  0
 3  2  5  6   0  1
```

Результаты расчета:

```
ip=1
x(1)=0.1428e+0000
x(2)=0.1142e+0001
x(4)=0.1142e+0001
x(6)=0.4285e+0000
f =-0.8857e+0001
```

Варианты заданий.

1 $f(\boldsymbol{x})=x_1 + x_2 \rightarrow \max\ (\min)$

$$\begin{cases} x_1 + 2x_2 \le 14 \\ -5x_1 - 3x_2 \le 15 \\ 2x_1 - 3x_2 \le 12 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

2 $f(\boldsymbol{x}) = 5x_1 + 4x_2 \rightarrow \max(\min)$

$$\begin{cases} x_1 - 2x_2 \le 6 \\ -x_1 + x_2 \le 8 \\ x_1 - x_2 \le 10 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

3 $f(\boldsymbol{x})=3x_1 + 2x_2 \rightarrow \max(\min)$

$$\begin{cases} x_1 + x_2 \le 6 \\ x_1 - 2x_2 \le 4 \\ x_1 - 3x_2 \le 3 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

4 $f(\boldsymbol{x})=5x_1 + 4x_2 \rightarrow \max(\min)$

$$\begin{cases} x_1 + x_2 \le 18 \\ 5x_1 - x_2 \le 20 \\ x_1 - x_2 \le 8 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

5 $f(\boldsymbol{x})=x_1 + 3x_2 \rightarrow \max\ (\min)$

$$\begin{cases} 2x_1 + x_2 \le 6 \\ -x_1 - 3x_2 \le 6 \\ x_1 + 2x_2 \le 8 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

6 $f(\boldsymbol{x})=2x_1+3x_2 \rightarrow \max(\min)$

$$\begin{cases} 2x_1 + x_2 \le 10 \\ -2x_1 + 3x_2 \le 6 \\ x_1 + x_2 \le 8 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

7 $f(\boldsymbol{x})=x_1 + 4x_2 \rightarrow \max\ (\min)$

$$\begin{cases} -x_1 + 5x_2 \le 20 \\ 3x_1 - x_2 \le 15 \\ x_1 + x_2 \le 6 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

8 $f(\boldsymbol{x})=6x_1 + 2x_2 \rightarrow \max(\min)$

$$\begin{cases} x_1 - x_2 \le 4 \\ -x_1 + 3x_2 \le 6 \\ x_1 - x_2 \le 3 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

9 $f(\boldsymbol{x})=5x_1 + 7x_2 \rightarrow \max\ (\min)$

$$\begin{cases} 5x_1 - 6x_2 \le 30 \\ -3x_1 + 14x_2 \le 42 \\ x_1 + 4x_2 \le 28 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

10 $f(\boldsymbol{x})=x_1 + 2x_2 \rightarrow \max\ (\min)$

$$\begin{cases} x_1 + 3x_2 \le 6 \\ -x_1 + x_2 \le 1 \\ 3x_1 - x_2 \le 6 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

11 $f(\boldsymbol{x})=3x_1-2x_2\to\max\ (\min)$

$$\begin{cases} x_1+2x_2\le 8 \\ -2x_1+x_2\le 2 \\ x_1-x_2\le 12 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

12 $f(\boldsymbol{x})=2x_1+3x_2\to\max\ (\min)$

$$\begin{cases} x_1+x_2\le 10 \\ -2x_1+3x_2\le 6 \\ x_1-x_2\le 4 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

13 $f(\boldsymbol{x})=3x_1+2x_2\to\max\ (\min)$

$$\begin{cases} 2x_1+x_2\le 8 \\ x_1+3x_2\le 6 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

14 $f(\boldsymbol{x})=x_1+4x_2\to\max\ (\min)$

$$\begin{cases} 2x_1+x_2\le 6 \\ x_1+3x_2\le 9 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

15 $f(\boldsymbol{x})=x_1+x_2\to\max\ (\min)$

$$\begin{cases} 3x_1+4x_2\le 12 \\ 2x_1-x_2\le 6 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

16 $f(\boldsymbol{x})=2x_1+3x_2\to\max\ (\min)$

$$\begin{cases} x_1+4x_2\le 12 \\ x_1+x_2\le 4 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

17 $f(\boldsymbol{x})=x_1+2x_2\to\max\ (\min)$

$$\begin{cases} 5x_1+4x_2\le 20 \\ 3x_1-x_2\le 6 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

18 $f(\boldsymbol{x})=x_1+x_2\to\max\ (\min)$

$$\begin{cases} x_1+2x_2\le 8 \\ 6x_1-x_2\le 3 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

19 $f(\boldsymbol{x})=x_1+5x_2\to\max\ (\min)$

$$\begin{cases} 2x_1+x_2\le 24 \\ -x_1+x_2\le 12 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

20 $f(\boldsymbol{x})=x_1+x_2\to\max\ (\min)$

$$\begin{cases} x_1+2x_2\le 14 \\ -5x_1+3x_2\le 15 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

21 $f(\boldsymbol{x})=x_1+x_2\to\max\ (\min)$

$$\begin{cases} x_1+2x_2\le 14 \\ 2x_1+x_2\le 10 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

22 $f(\boldsymbol{x})=2x_1+3x_2\to\max\ (\min)$

$$\begin{cases} -4x_1+2x_2\le 4 \\ x_1+x_2\le 6 \end{cases}$$

$$x_1\ge 0, x_2\ge 0$$

23 $f(\boldsymbol{x})=2x_1 + x_2 \rightarrow$ max (min)

$$\begin{cases} 4x_1 + x_2 \le 16 \\ x_1 + x_2 \le 11 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

24 $f(\boldsymbol{x})=2x_1 + x_2 \rightarrow$ max (min)

$$\begin{cases} x_1 + x_2 \le 3 \\ x_1 - 2x_2 \le 1 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

25 $f(\boldsymbol{x})=x_1 + 6x_2 \rightarrow$ max (min)

$$\begin{cases} x_1 + x_2 \le 10 \\ x_1 \le 8 \\ -x_1 + x_2 \le 3 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

26 $f(\boldsymbol{x})=x_1 + 2x_2 \rightarrow$ max (min)

$$\begin{cases} 2x_1 - x_2 \le 6 \\ -x_1 + 3x_2 \le 6 \\ x_1 + x_2 \le 4 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

27 $f(\boldsymbol{x})=6x_1 + x_2 \rightarrow$ max (min)

$$\begin{cases} x_1 + x_2 \le 20 \\ -x_1 + x_2 \le 15 \\ x_1 - 3x_2 \le 9 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

28 $f(\boldsymbol{x})=x_1 + 4x_2 \rightarrow$ max (min)

$$\begin{cases} x_1 - 3x_2 \le 6 \\ x_1 + x_2 \le 9 \\ -x_1 + x_2 \le 4 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

29 $f(\boldsymbol{x})=x_1 + x_2 \rightarrow$ max (min)

$$\begin{cases} 3x_1 + 4x_2 \le 12 \\ 4x_1 - x_2 \le 8 \\ 2x_2 \le 12 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

30 $f(\boldsymbol{x})=2x_1 + x_2 \rightarrow$ max (min)

$$\begin{cases} x_1 + 2x_2 \le 10 \\ -2x_1 + x_2 \le 2 \\ x_1 - 8x_2 \le 8 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

6.2 Лабораторная работа №2. Решение двойственной задачи линейного программирования

1. Составить задачу двойственную к основной;
2. Решить двойственную задачу симплекс - методом (см.лабораторную работу №1, программа simplecs);
3. Сделать анализ.

Пример.

$$f(\mathbf{x})=10x_1+14x_2+12x_3 \to \max;$$

$$\begin{cases} 4x_1 + 2x_2 + x_3 \le 180 \\ 3x_1 + x_2 + 3x_3 \le 210 \\ x_1 + 2x_2 + 5x_3 \le 244 \end{cases}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0.$$

Двойственная задача по отношению к исходной:

$$f^*(\mathbf{y})=180y_1+210y_2+244y_3 \to \min;$$

$$\begin{cases} 4y_1 + 3y_2 + y_3 \ge 10 \\ 2y_1 + y_2 + 2y_3 \ge 14 \\ y_1 + 3y_2 + 5y_3 \ge 12 \end{cases}$$

$$y_1 \ge 0, y_2 \ge 0, y_3 \ge 0.$$

Результаты расчета:

$$y_1^* = 5.75$$

$$y_2^* = 0$$

$$y_3^* = 1.25$$

$$f_{\min}^* = 1340.$$

Анализ результатов.

Здесь y_1^*, y_3^*, обозначают условные двойственные оценки единицы сырья 1-го и 3-го видов, отличные от 0. По оценкам можно судить, что сырье 1-го и 3-го видов полностью используется при оптимальном плане производства продукции.

Оценка $y_2^*=0$, поэтому 2-ой вид сырья не полностью используется при оптимальном плане производства продукции.

Подставим оптимальные двойственные оценки в систему ограничений двойственной задачи:

23+1,25>10

11,5+2,5=14

5,75+6,25=12.

Первое ограничение выполняется как строгое неравенство. Это означает, что двойственная оценка сырья, используемого на производство одного изделия 1-го вида, выше цены этого изделия и выпускать его невыгодно.

Варианты заданий.

1 $f(\boldsymbol{x})=x_1+2x_2 \to \max (\min)$

$$\begin{cases} 2x_1 - x_2 \ge 6 \\ 2x_1 + x_2 \le 1 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

2 $f(\boldsymbol{x})=x_1+3x_2 \to \max (\min)$

$$\begin{cases} -7x_1 + 4x_2 \ge 28 \\ x_1 - 3x_2 \ge 15 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

3 $f(\boldsymbol{x})=x_1+3x_2 \to \max (\min)$

$$\begin{cases} -x_1 + x_2 \ge 6 \\ x_1 - 2x_2 \ge 10 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

4 $f(\boldsymbol{x})=2x_1+3x_2 \to \max (\min)$

$$\begin{cases} x_1 - 4x_2 \ge 12 \\ -4x_1 + x_2 \ge 4 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

5 $f(\boldsymbol{x})=x_1+x_2 \to \max (\min)$

$$\begin{cases} x_1 - 2x_2 \ge 14 \\ -5x_1 + 3x_2 \ge 15 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

6 $f(\boldsymbol{x})=3x_1+x_2 \to \max (\min)$

$$\begin{cases} -7x_1 + 3x_2 \ge 21 \\ x_1 - 5x_2 \ge 10 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

7 $f(\boldsymbol{x})=5x_1+7x_2 \to \max (\min)$

$$\begin{cases} 5x_1 - 6x_2 \le 30 \\ x_1 - 4x_2 \le 28 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

8 $f(\boldsymbol{x})=3x_1+x_2 \to \max (\min)$

$$\begin{cases} x_1 - 2x_2 \ge 6 \\ x_1 + 5x_2 \ge 8 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

9 $f(\boldsymbol{x})=3x_1+2x_2 \to \max\ (\min)$

$$\begin{cases} 2x_1+x_2 \ge 12 \\ x_1+x_2 \le 1 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

10 $f(\boldsymbol{x})=2x_1+x_2 \to \max(\min)$

$$\begin{cases} -2x_1+x_2 \ge 10 \\ x_1-x_2 \ge 8 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

11 $f(\boldsymbol{x})=3x_1+x_2 \to \max\ (\min)$

$$\begin{cases} -x_1-2x_2 \ge 8 \\ -2x_1+x_2 \le 2 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

12 $f(\boldsymbol{x})=5x_1+4x_2 \to \max\ (\min)$

$$\begin{cases} x_1-5x_2 \ge 20 \\ -x_1+x_2 \ge 9 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

13 $f(\boldsymbol{x})=x_1+x_2 \to \max\ (\min)$

$$\begin{cases} -2x_1-x_2 \ge 6 \\ -x_1+3x_2 \le 9 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

14 $f(\boldsymbol{x})=x_1+x_2 \to \max\ (\min)$

$$\begin{cases} 3x_1-4x_2 \ge 12 \\ x_1+x_2 \le 1 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

15 $f(\boldsymbol{x})=-2x_1-3x_2 \to \max\ (\min)$

$$\begin{cases} -4x_1+2x_2 \ge 4 \\ x_1-x_2 \ge 6 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

16 $f(\boldsymbol{x})=x_1-x_2 \to \max\ (\min)$

$$\begin{cases} 2x_1-3x_2 \ge 12 \\ x_1+x_2 \le 1 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

17 $f(\boldsymbol{x})=x_1+3x_2 \to \max\ (\min)$

$$\begin{cases} 3x_1+4x_2 \le 12 \\ 2x_1-x_2 \ge 10 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

18 $f(\boldsymbol{x})=6x_1+x_2 \to \max\ (\min)$

$$\begin{cases} -x_1+x_2 \ge 15 \\ x_1-3x_2 \ge 9 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

19 $f(\boldsymbol{x})=x_1+2x_2 \to \max\ (\min)$

$$\begin{cases} x_1-6x_2 \ge 6 \\ -2x_1+x_2 \ge 2 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

20 $f(\boldsymbol{x})=4x_1+x_2 \to \max\ (\min)$

$$\begin{cases} -6x_1+x_2 \ge 12 \\ x_1+2x_2 \le 10 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

21 $f(\boldsymbol{x})=x_1+5x_2 \to \max\ (\min)$

$$\begin{cases} -x_1-x_2 \ge 12 \\ x_1-4x_2 \le 8 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

22 $f(\boldsymbol{x})=2x_1+x_2 \to \max\ (\min)$

$$\begin{cases} x_1-5x_2 \ge 10 \\ x_1-2x_2 \le 6 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

23 $f(\boldsymbol{x})=2x_1+3x_2\to\max\ (\min)$

$$\begin{cases} x_1+x_2\ge 10 \\ 2x_1+x_2\le 6 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

24 $f(\boldsymbol{x})=3x_1+2x_2\to\max(\min)$

$$\begin{cases} 4x_1+3x_2\le 8 \\ x_1+x_2\ge 4 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

25 $f(\boldsymbol{x})=x_1+4x_2\to\max\ (\min)$

$$\begin{cases} 2x_1-2x_2\ge 14 \\ -x_1-x_2\ge 6 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

26 $f(\boldsymbol{x})=x_1+3x_2\to\max\ (\min)$

$$\begin{cases} x_1-2x_2\ge 6 \\ -x_1+3x_2\ge 6 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

27 $f(\boldsymbol{x})=2x_1+x_2\to\max\ (\min)$

$$\begin{cases} x_1+x_2\ge 2 \\ x_1+2x_2\ge 8 \\ 3x_1-x_2\le 6 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

28 $f(\boldsymbol{x})=6x_1+2x_2\to\max\ (\min)$

$$\begin{cases} -3x_1+x_2\ge 6 \\ -3x_1+x_2\le 1 \\ x_1+4x_2\le 28 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

29 $f(\boldsymbol{x})=5x_1+2x_2\to\max\ (\min)$

$$\begin{cases} x_1-3x_2\ge 9 \\ 3x_1-x_2\le 6 \\ 2x_2\le 12 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

30 $f(\boldsymbol{x})=2x_1+x_2\to\max\ (\min)$

$$\begin{cases} -x_1+x_2\ge 1 \\ x_1-2x_2\ge 1 \\ x_1-x_2\le 8 \\ x_1\ge 0, x_2\ge 0 \end{cases}$$

6.3 Лабораторная работа №3. Решение транспортной задачи

1. Нахождение опорного плана методом северо-западного угла или минимального элемента.
2. Нахождение оптимального решения методом потенциалов.

Входные данные:

– Метод нахождения опорного плана(1-северо-западного угла,2-минимального элемента);
– Выбор ввода (y -из файла;n- формируется таблица с клавиатуры);
– M - количество пунктов отправления;
– N- количество пунктов потребления;
– A_i –запасы в i-ом пункте ;
– B_j-потребность j-го пункта;
– $C_{i\,j}$-элементы матрицы стоимостей перевозок единицы продукции.

Выходные данные:

– Таблицы с циклами пересчета;
– Таблица оптимального плана грузоперевозок;
– *f*-минимальная стоимость перевозок

Пример.

$A_1=140$ $B_1=60$
$A_2=180$ $B_2=70$
$A_3=160$ $B_3=120$
$B_4=130$
$B_5=100$

$$C = \begin{pmatrix} 2 & 3 & 4 & 2 & 4 \\ 8 & 5 & 1 & 4 & 1 \\ 9 & 8 & 4 & 7 & 2 \end{pmatrix}$$

Текст программы.

```
{This is Transport Task for Open and close task together}
Uses Crt;
Label l1;
Const N=8;
    n1=7;
```

```
      n2=7;
      Sa:longint=0;
      Sb:longint=0;
      AColor=cyan;
      BColor=cyan;
      CColor=LightRed;
      UColor=LightGreen;
      VColor=LightGreen;
      PColor=yellow;
      Allcolor=White;
      TableColor=White;
      TableTextColor=White;
      ErrColor=LightRed;
      backGround=1;
Type predpr=Array [1..N] of longint;
     rasp=Array [1..N,1..N] of longint;
Var A,B,U_potenc,V_potenc,B_d,x:predpr;
    c,p:rasp;
    f,f0,x_min,Sp:longint;
    Nt,x_p,r,r_min,ki,kj,Na,Nb,h,l,i,j:byte;
    d:char;
    u:Array[1..N*N] of byte;
    method:byte;
Procedure ZeroArray (var a:predpr);
var i:byte;
Begin
     for i:=1 to N do a[i]:=0;
End;

Procedure WriteXYC (x,y:byte; s:string; c:byte);
Begin
     TextColor(c);
     GotoXY(x,y);
     Write(s);
End;
Procedure WriteXYF (x,y:byte; n:byte; a:longint; c:byte);
Begin
     TextColor(c);
     GotoXY(x,y); Write(' ':n);
     GotoXY(x,y); Write(a);
End;
Procedure InputVar (var x:longint; y:byte);
var i:integer;
    s:string;
    c:char;
```

```
   j,k:byte;
Begin
    s:=''; i:=1;
    TextColor(AllCOlor);
    Repeat
        c:=ReadKey;
        Case ord(c) of
48..57:         begin s:=s+c;
                  Write(c);
                  inc(i);
              end;
8:             if i>1 then begin dec(i);
                  Delete(s,i,1);
                  Write(chr(8),' ',chr(8));
              end;
        end;
        j:=WhereX;
        GotoXY(60,1); ClrEOL;
        if i>y then begin
          TextColor(ErrColor);
          Write('ЌЃ Ў®«ЃЃ ');
          for k:=1 to y-1 do Write('9');
          TextColor(AllCOlor);
        end;
        GotoXY(j,1);
    Until (ord(c)=13) and (i<y+1);
    val(s,x,i);
End;
Procedure HorizLine (a,b,c,d,e:char);
var i,j:byte;
Begin
    Write(a);
    for i:=1 to n2 do Write(b);
    Write(c);
    for i:=1 to Nb do begin
       for j:=1 to n1 do Write(b);
       if i<>Nb then Write(d) else Write(c);
    end;
    for i:=1 to 4 do Write(b);
    Write(e);
End;
Procedure VertLine;
var i:byte;
Begin
    Write('i',' ':n2,'i');
```

```
        for i:=1 to Nb-1 do Write(' ':n1,'i');
    WriteLn(' ':n1,'i',' ' :4,'i');
End;
procedure ClearWind(x1,y1,x2,y2:byte);
var i,j:byte;
begin
for i:=x1 to x2 do
 for j:=y1 to y2 do
  begin
   gotoxy(i,j);
   Write(#32);
  end;
end;
Procedure DrawTable;
Begin
{    ClrScr;}
   ClearWind(1,1,Lo(windmax),na*5);
   TextColor(TableCOlor);
   h:=6+Na*3;
   l:=14+Nb*7;
   GotoXY(1,3);
   for i:=3 to h do VertLine;
   GotoXY(1,2);
   HorizLine('+','Д','+','+','+');
   for i:=1 to Na+1 do begin
      GotoXY(1,i*3+2);
      if (i=1) or (i=Na+1)
        then HorizLine('+','H','+','+','+')
        else HorizLine('+','Д','+','+','+');
   end;
   GotoXY(1,h+1);
   HorizLine('+','Д','+','+','+');
   TextColor(TableTextColor);
   for i:=1 to Na do begin
      GotoXY(5,i*3+3);
      Write('A',i);
   end;
   for i:=1 to Nb do begin
      GotoXY(i*(n1+1)+n2-2,3);
      Write('B',i);
   end;
   l:=Nb*(n1+1)+n2+3;
   h:=Na*3+6;
   WriteXYC(4,3,'B-->',TableTextColor);
   WriteXYC(4,4,'□□A',TableTextColor);
```

```
{    WriteXYC(1,1,'' Ў«Ёж  N1',AllColor);}
    WriteXYC(l,4,' U',TableTextColor);
    WriteXYC(3,h,'  V',TableTextColor);
End;
Procedure InputC (var a:predpr; b:byte; c:char);
var i,l,m:byte;
Begin
    for i:=1 to b do begin
        TextColor(AllColor);
        GotoXY(32,1);
        ClrEOL;
        Write(c,i,'=  ');
        InputVar(a[i],n1);
        TextColor(CColor);
        Case c of
'A':    GotoXY(n2-trunc(ln(a[i])/ln(10)),i*3+4);
'B':    GotoXY(n2+i*(n1+1)-trunc(ln(a[i])/ln(10)),4);
        end;
        Write(a[i]);
    end;
End;
Function PricePlan:longint;
var i,j:byte;
   f:longint;
Begin
    f:=0;
    for i:=1 to Na do
        for j:=1 to Nb do
            if p[i,j]>0 then inc(f,c[i,j]*p[i,j]);
    GotoXY(2,Hi(windMax)-2);
    ClrEOL;
    TextColor(PColor);
    Write('Func = ',f);
    PricePlan:=f;
End;

Function CalcPotenc:boolean;
var k,i,j:byte;          {U_potenc Ё V_potenc}
   Z_a,Z_b:predpr;
   d:boolean;
Begin
    ZeroArray(Z_a); ZeroArray(Z_b);
    U_potenc[1]:=0; Z_a[1]:=1; k:=1;
    Repeat
        d:=1=1;
```

```
for i:=1 to Na do
            if Z_a[i]=1 then
              for j:=1 to Nb do
                if (p[i,j]>-1) and (Z_b[j]=0) then begin
                  Z_b[j]:=1;
                  V_potenc[j]:=c[i,j]-U_potenc[i];
                  inc(k);
                  d:=1=2;
                end;
          for i:=1 to Nb do
            if Z_b[i]=1 then
              for j:=1 to Na do
                if (p[j,i]>-1) and (Z_a[j]=0) then begin
                  Z_a[j]:=1;
                  U_potenc[j]:=c[j,i]-V_potenc[i];
                  inc(k);
                  d:=1=2;
                end;
    Until (k=Na+Nb) or d;
    if d then begin
      i:=1;
      While Z_a[i]=1 do inc(i);
      j:=1;
      While Z_b[j]=0 do inc(j);
      p[i,j]:=0;
      WriteXYF((j+1)*(n1+1)+n2-8,i*3+4,1,p[i,j],7);
    end;
for i:=1 to n1 do U_Potenc[i]:=-U_Potenc[i];
    CalcPotenc:=d;
End;
Procedure OutPlan;
var i,j,h,l,k:byte;
   c_max:longint;
Begin
    k:=0;
    for i:=1 to Na do begin
       h:=i*3+4;
       for j:=1 to Nb do begin
          l:=j*(n1+1)+n2-5;
          GotoXY(l,h);
          Write(' ':n1);
          if p[i,j]>0 then begin
            inc(k);
            WriteXYF(l-trunc(ln(p[i,j])/ln(10))+5,h,1,p[i,j],14);
          end
```

```
            else if p[i,j]=0 then begin
                    WriteXYF(l+n1-2,h,1,p[i,j],14);
                    inc(k);
                end;
            end;
        end;
        While CalcPotenc do inc(k);
        if k>Na+Nb-1 then WriteXYC(40,1,'k > n+m-1',ErrColor);
End;
Function CalcPotecTable(var ki,kj:byte):integer;
var i,j:byte;
    k,k_min:integer;
    b:boolean;
Begin
    b:=1=1;
    for i:=1 to Na do
        for j:=1 to Nb do
            if p[i,j]=-1 then begin
              k:=c[i,j]+U_potenc[i]-V_potenc[j];
              if b then begin
                 b:=1=2;
                 ki:=i; kj:=j; k_min:=k;
              end else
                  if k<k_min then begin
                     k_min:=k;
                     ki:=i; kj:=j;
                  end;
              TextColor(ErrColor);
              GotoXY(j*(n1+1)+n2-5,i*3+4);
              Write('(',k,')');
            end;
    if k_min<0 then WriteXYC(kj*(n1+1)+n2,ki*3+4,'X',ErrColor);
    CalcPotecTable:=k_min;
End;
Procedure Array1Dto2D(c:byte; var a,b:byte);
Begin
    b:=c mod Nb; a:=c div Nb +1;
    if b=0 then begin
       b:=Nb; dec(a);
    end;
End;
Procedure CalcContur(Xi,Yi:byte; var z:boolean; var c:byte);
var i,j:byte;
Begin
  z:=1=2;
```

```
  Case c of
1:  for i:=1 to Na do
      if i<>Xi then
        if p[i,Yi]>-1 then begin
          if u[(i-1)*Nb+Yi]=0 then begin
            u[(Xi-1)*Nb+Yi]:=(i-1)*Nb+Yi;
            c:=2;
            CalcContur(i,Yi,z,c);
            if z then exit;
          end;
        end
        else if (i=ki) and (Yi=kj) then begin
              u[(Xi-1)*Nb+Yi]:=(ki-1)*Nb+kj;
              z:=not z;
              exit;
        end;
2:  for i:=1 to Nb do
      if i<>Yi then
        if p[Xi,i]>-1 then begin
          if u[(Xi-1)*Nb+i]=0 then begin
            u[(Xi-1)*Nb+Yi]:=(Xi-1)*Nb+i;
            c:=1;
            CalcContur(Xi,i,z,c);
            if z then exit;
          end;
        end
        else if (Xi=ki) and (i=kj) then begin
              u[(Xi-1)*Nb+Yi]:=(ki-1)*Nb+kj;
              z:=not z;
              exit;
        end;
  end;
  u[(Xi-1)*Nb+Yi]:=0;
  c:=c mod 2 +1;
End;

Procedure OutContur;
var i,j,k,mi,mj,l:byte;
   z:boolean;
   p_m:longint;
Begin
    for i:=1 to N*N do u[i]:=0;
    l:=1;
    CalcContur(ki,kj,z,l);
    i:=ki; j:=kj;
```

```
    k:=u[(i-1)*Nb+j];
    Array1Dto2D(k,i,j);
    mi:=i; mj:=j; l:=1;
    Repeat
        inc(l);
        k:=u[(i-1)*Nb+j];
        Array1Dto2D(k,i,j);
        if l mod 2=1 then
          if p[i,j]<p[mi,mj] then begin
            mi:=i; mj:=j;
          end;
    Until (i=ki) and (j=kj);

    i:=ki; j:=kj; l:=0;
    p_m:=p[mi,mj];
    Repeat
        if l mod 2=0 then begin
          inc(p[i,j],p_m);
          WriteXYC((n1+1)*j+n2-1,i*3+3,'(+)',Errcolor);
        end else begin
           dec(p[i,j],p_m);
           WriteXYC((n1+1)*j+n2-1,i*3+3,'(-)',errcolor);
        end;
        if l=0 then inc(p[i,j]);
        k:=u[(i-1)*Nb+j];
        Array1Dto2D(k,i,j);
        inc(l);
    Until (i=ki) and (j=kj);
    p[mi,mj]:=-1;
End;
Procedure PressAnyKey;
var d:char;
Begin
    TextColor(AllCOlor);
    GotoXY(40,1);
    Write(' (SPACE)');
    while ReadKey<>#32 do;
    GotoXY(40,1);
    ClrEOL;
End;
 var ft:text;bt:boolean;cp:char;
BEGIN
TextBackGround(background);
   ClrScr;
   ZeroArray(U_potenc); ZeroArray(V_potenc);
```

```
    Nt:=1;
    TextColor(AllColor);
    method:=0;
    repeat
    write('');
    readln(Na);
    if (na<>1)and(na<>2) then method:=0 else method:=na;
    until method<>0;
repeat
write(' [Y/N]: ');
readln(cp);
until (UpCase(cp)='Y')or(UpCase(cp)='N');
if UpCase(cp)='Y' then bt:=true else bt:=false;
 if not bt then begin{bt}
    Repeat
         Write(', ',N-1,':   ');
         ReadLn(Na);
         Write(', ',N-1,'   ');
         ReadLn(Nb);
    Until (Na>1) and (Na<=N-1) and (Nb>1) and (Nb<=N-1);

if na>5 then TextMode(C80 + Font8x8);

TextBackGround(background);
    ClrScr;
    DrawTable;
    WriteXYC(1,1,',,AllColor);
    InputC(A,Na,'A');
    InputC(B,Nb,'B');
    TextColor(AllColor);
    GotoXY(1,1); ClrEOL;
    Write(',);
    for i:=1 to Na do
       for j:=1 to Nb do begin
          TextColor(AllCOlor);
          GotoXY(29,1); ClrEOL;
          Write('A',i,' - B',j,'  ');
          InputVar(c[i,j],5);
          WriteXYF((n1+1)*j+n2-4,i*3+3,1,c[i,j],11);
       end;
      END{BT} else
      begin{prev}
       assign(ft,'prev.dat');
       {$I-}
       reset(ft);
```

```
        {$I+}
        if IOResult<>0 then
         begin
          writeln('Error to open file prev.dat');
          exit;
         end;
        readln(ft,na,nb);
        for i:=1 to na do read(ft,a[i]);
        readln(ft);
        for i:=1 to nb do
         begin
         read(ft,b[i]);
         end;
        readln(ft);
   if na>5 then TextMode(C80 + Font8x8);
TextBackGround(background);
    ClrScr;
    DrawTable;
        for i:=1 to na do
         begin
        for j:=1 to nb do
         begin
         read(ft,c[i,j]);
            TextColor(AllCOlor);
                  WriteXYF((n1+1)*j+n2-4,i*3+3,1,c[i,j],11);
         end;
        readln(ft);
         end;
         close(ft);
       end;
(*************************************************************)
    GotoXY(1,1);
    ClrEOL;
    TextColor(AllCOlor);
 {   Write('' Ў«Ёж  N1');}
    for i:=1 to Na do Sa:=Sa+A[i];
    for i:=1 to Nb do Sb:=Sb+B[i];
    if Sa<>Sb then begin
       WriteXYC(20,1,',AllColor);
       d:=ReadKey;
       if Sa>Sb then begin
          inc(Nb);
          B[Nb]:=Sa-Sb;
          WriteXYC(1,Hi(windMax)-1,',,,AllCOlor);
          for i:=1 to Na do c[i,Nb]:=0;
```

```
      end else begin
         inc(Na);
         A[Na]:=Sb-Sa;
         for i:=1 to Nb do c[Na,i]:=0;
         WriteXYC(1,Hi(windMax)-1,',,',AllColor);
      end;
      DrawTable;
      for i:=1 to Na do
         for j:=1 to Nb do WriteXYF((n1+1)*j+n2-4,i*3+3,1,c[i,j],11);
      for i:=1 to Na do
         WriteXYF(n2-trunc(ln(A[i])/ln(10)),i*3+4,1,A[i],14);
      for i:=1 to Nb do
         WriteXYF(n2+i*(n1+1)-trunc(ln(B[i])/ln(10)),4,1,B[i],14);
      WriteXYC(20,1,',AllColor);
   end
      else WriteXYC(20,1, ',AllColor);
assign(ft,'prev.dat');
rewrite(ft);
  writeln(ft,na,' ',nb);
for i:=1 to na do write(ft,a[i],' ');
writeln(ft);
for i:=1 to nb do write(ft,b[i],' ');
writeln(ft);
for i:=1 to na do
 begin
  for j:=1 to nb do
   write(ft,c[i,j],' ');
   writeln(ft);
 end;
close(ft);
case method of
2:{Min element}BEGIN
   for i:=1 to Nb do B_d[i]:=B[i];
   for i:=1 to Na do begin
      for j:=1 to Nb do x[j]:=j;
      for j:=1 to Nb-1 do begin
         x_min:=c[i,x[j]];
         r_min:=j;
         for r:= j+1 to Nb do
            if (x_min>c[i,x[r]]) or
             ((x_min=c[i,x[r]]) and (B[x[r]]>b[x[r_min]])) then
            begin
               x_min :=c[i,x[r]];
               r_min:=r;
            end;
```

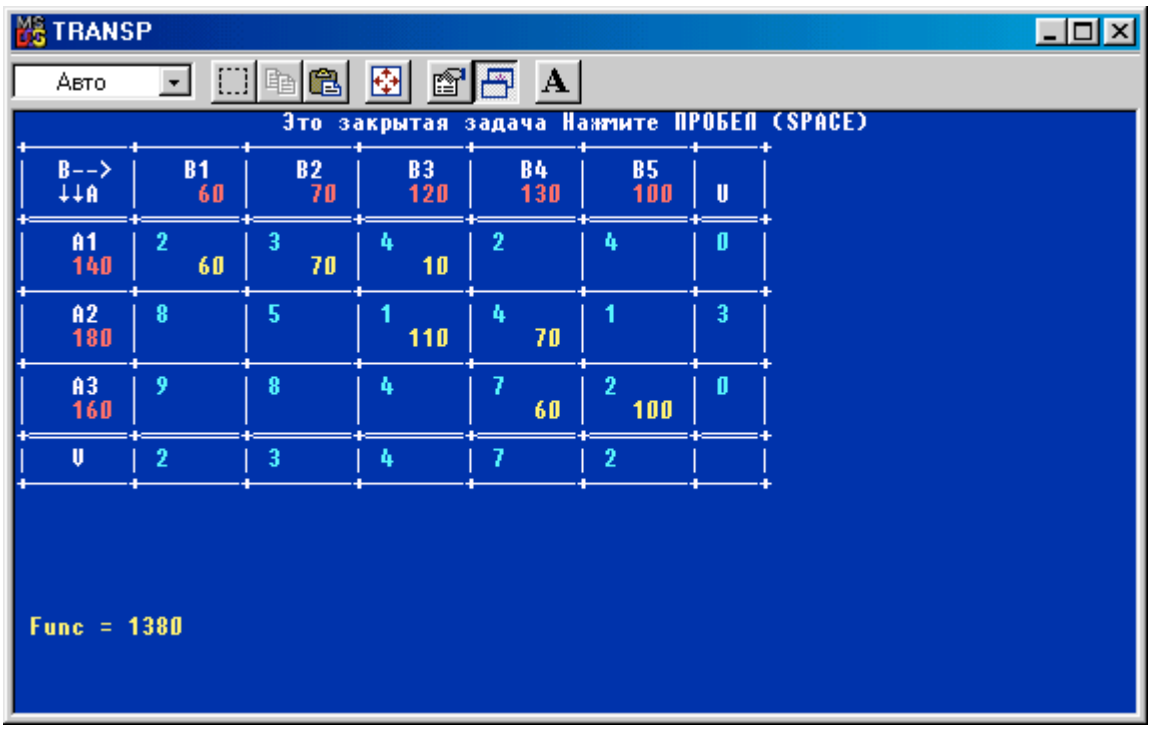
```
            x_p:=x[r_min];
            x[r_min]:=x[j];
            x[j]:=x_p;
         end;
         Sp:=0;
         for j:=1 to Nb do begin
            p[i,x[j]]:=B_d[x[j]];
            if p[i,x[j]]>A[i]-Sp then p[i,x[j]]:=A[i]-Sp;
            inc(Sp,p[i,x[j]]);
            dec(B_d[x[j]],p[i,x[j]]);
         end;
      end;
   END;
   1:{Noth-West element}BEGIN
      for i:=1 to Nb do B_d[i]:=B[i];
      for i:=1 to Na do begin
         for j:=1 to Nb do x[j]:=j;
         for j:=1 to Nb-1 do begin
            x_min:=c[i,x[j]];
            r_min:=j;
         end;
         Sp:=0;
         for j:=1 to Nb do begin
            p[i,x[j]]:=B_d[x[j]];
            if p[i,x[j]]>A[i]-Sp then p[i,x[j]]:=A[i]-Sp;
            inc(Sp,p[i,x[j]]);
            dec(B_d[x[j]],p[i,x[j]]);
         end;
      end;
   END;
   end;{case method}
   (*****************************************************)
      for i:=1 to Na do
         for j:=1 to Nb do if p[i,j]=0 then p[i,j]:=-1;
      OutPlan;
      f:=PricePlan; f0:=F;
      While CalcPotenc do;
      for i:=1 to Na do WriteXYF(l+1,i*3+3,3,U_potenc[i],11);
      for i:=1 to Nb do WriteXYF(i*(n1+1)+n2-4,h,6,V_potenc[i],11);
      PressAnyKey;

      While CalcPotecTable(ki,kj)<0 do begin
           OutContur;
           PressAnyKey;
           for i:=1 to Na do
```

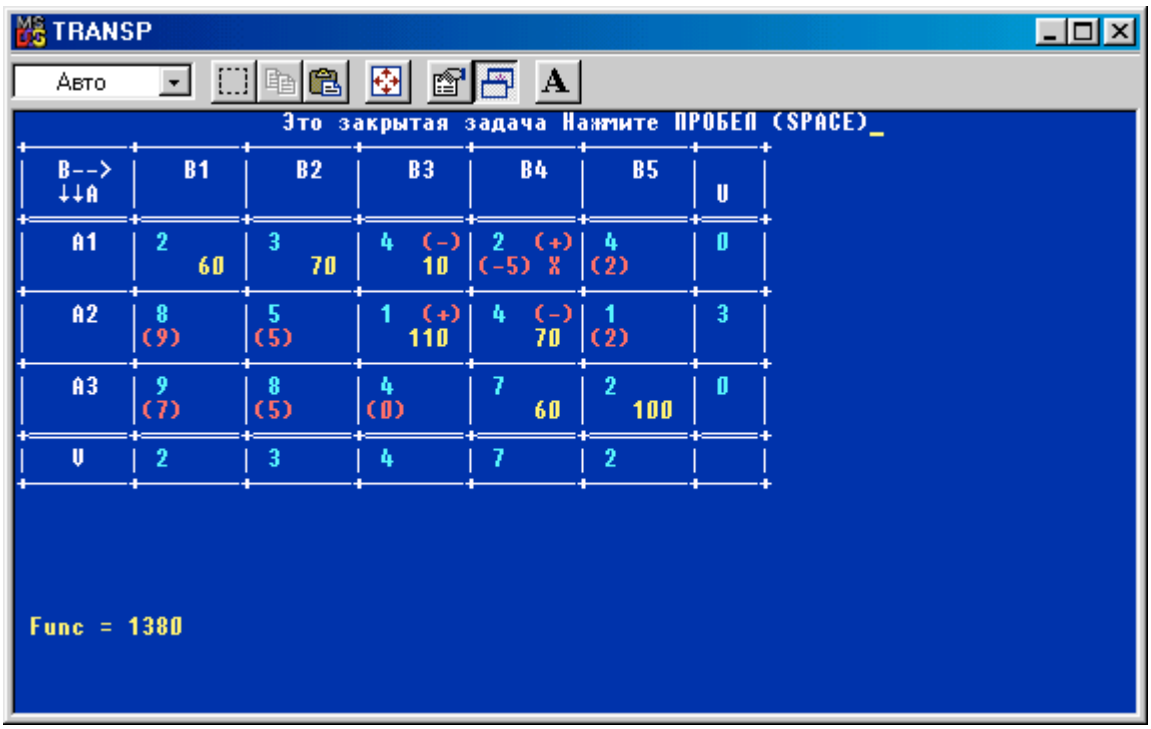
```
          for j:=1 to Nb do WriteXYC((n1+1)*j+n2-1,i*3+3,'   ',AllColor);
        inc(Nt);
        GotoXY(1,1);
{       Write('' Ў«Ёж  N',Nt);}
        OutPlan;
        f0:=f; f:=PricePlan;
        if CalcPotenc then Goto l1;
        for i:=1 to Na do WriteXYF(l+1,i*3+3,3,U_potenc[i],11);
        for i:=1 to Nb do WriteXYF(i*(n1+1)+n2-4,h,6,V_potenc[i],11);
        PressAnyKey;
   end;
(*********************************************************)
   WriteXYC(40,1,,ErrColor);
   WriteXYC(60,1,',AllCOlor);
   for i:=1 to Na do
      for j:=1 to Nb do if p[i,j]=-1 then begin
         h:=i*3+4;
         l:=j*(n1+1)+n2-5;
         GotoXY(l,h);
         Write(' ':n1);
      end;
   GotoXY(40,1);
l1: d:=ReadKey;
TextMode(lastmode);
END.
```

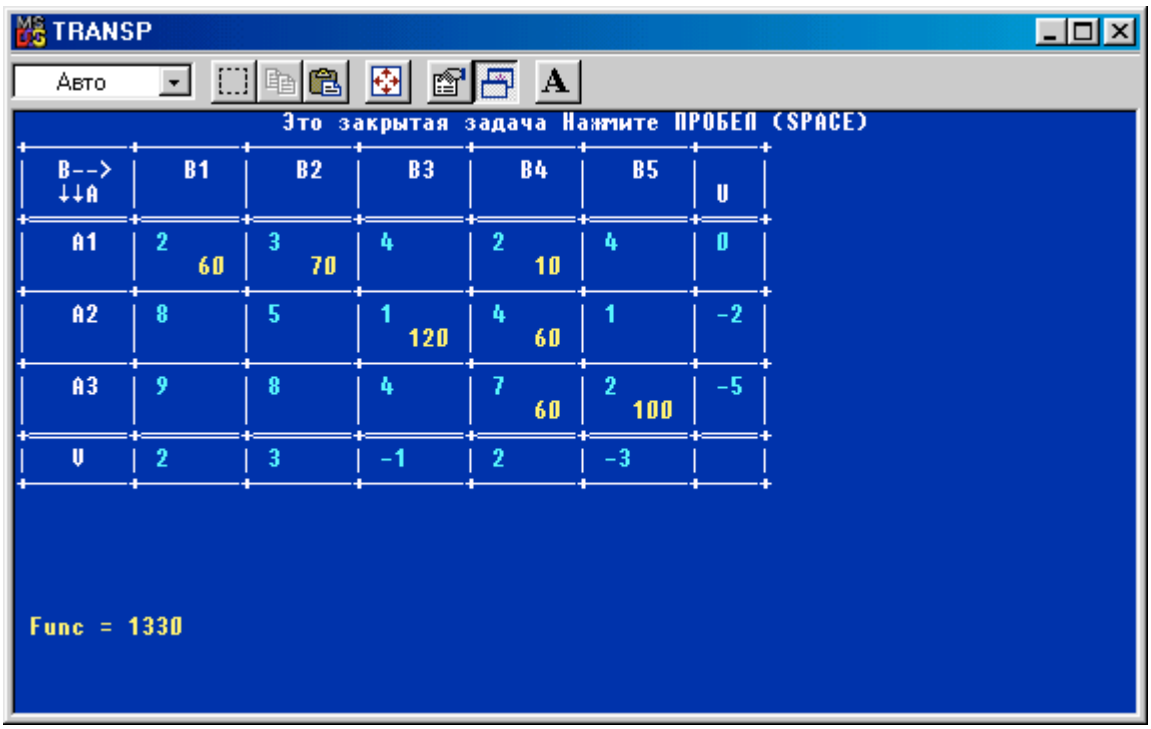
Входные данные:

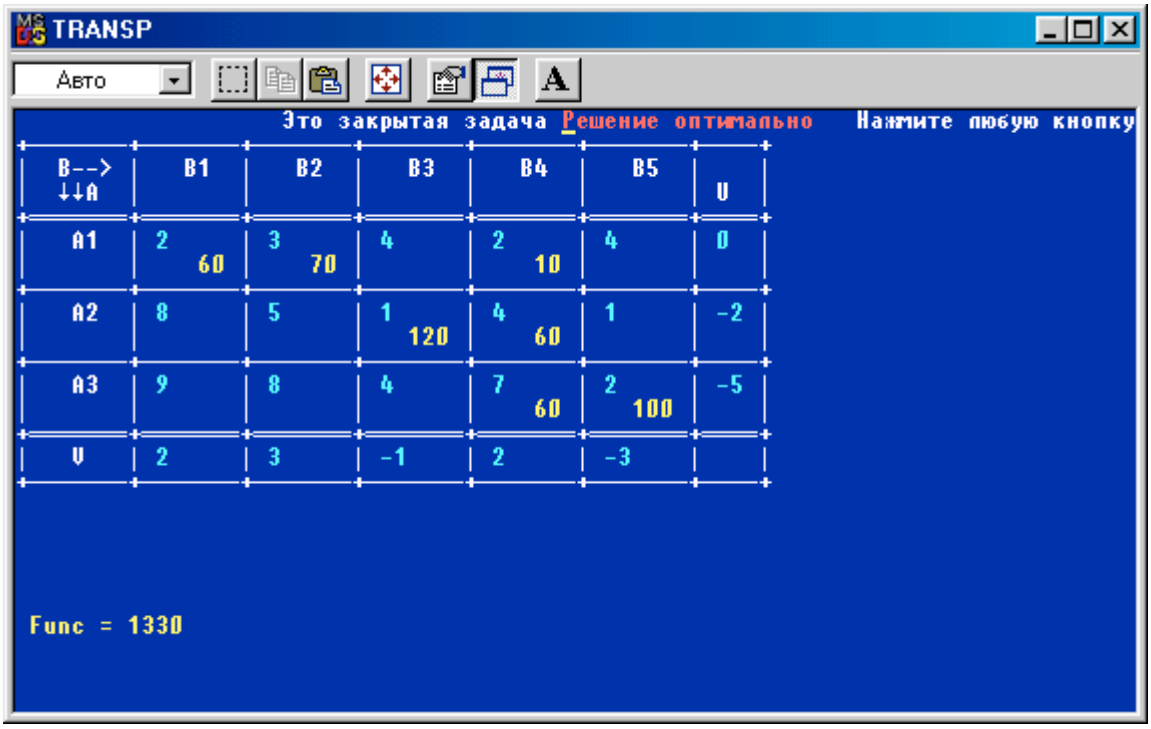
Введите метод (1–метод северо-западного угла ;
2–метод минимального элемента) 1
Введите запасы A_1 –A_3
Введите потребности B_1 –B_5
Введите матрицу стоимости перевозок C из A_i в B_j
Формируется таблица входных данных.



Результат выполнения программы.




TRANSP
Авто
Это закрытая задача Нажмите ПРОБЕЛ (SPACE)
B--> ↓↓A | B1 | B2 | B3 | B4 | B5 | U
A1 | 2 60 | 3 70 | 4 | 2 10 | 4 | 0
A2 | 8 | 5 | 1 120 | 4 60 | 1 | -2
A3 | 9 | 8 | 4 | 7 60 | 2 100 | -5
V | 2 | 3 | -1 | 2 | -3 |
Func = 1330


TRANSP
Авто
Это закрытая задача Решение оптимально Нажмите любую кнопку
B--> ↓↓A | B1 | B2 | B3 | B4 | B5 | U
A1 | 2 60 | 3 70 | 4 | 2 10 | 4 | 0
A2 | 8 | 5 | 1 120 | 4 60 | 1 | -2
A3 | 9 | 8 | 4 | 7 60 | 2 100 | -5
V | 2 | 3 | -1 | 2 | -3 |
Func = 1330

Варианты заданий.

1) a_1 =200 b_2 = 130
a_2 =175 b_3 = 80
a_3 =225 b_4 = 190
b_1 = 100 b_5 = 100

$$D = \begin{pmatrix} 5 & 7 & 4 & 2 & 5 \\ 7 & 1 & 3 & 1 & 10 \\ 2 & 3 & 6 & 8 & 7 \end{pmatrix}$$

2) a_1 =200 b_2 = 125
a_2 =450 b_3 = 325
a_3 =250 b_4 = 250
b_1 = 100 b_5 = 100

$$D = \begin{pmatrix} 5 & 8 & 7 & 10 & 3 \\ 4 & 2 & 2 & 5 & 6 \\ 7 & 3 & 5 & 9 & 2 \end{pmatrix}$$

3) a_1 =250 b_2 = 130
a_2 =200 b_3 = 100
a_3 =200 b_4 = 160
b_1 = 120 b_5 = 140

$$D = \begin{pmatrix} 27 & 36 & 35 & 31 & 29 \\ 22 & 23 & 26 & 32 & 35 \\ 35 & 42 & 38 & 32 & 39 \end{pmatrix}$$

4) a_1 =350 b_2 = 170
a_2 =330 b_3 = 220
a_3 =270 b_4 = 150
b_1 = 210 b_5 = 200

$$D = \begin{pmatrix} 3 & 12 & 9 & 1 & 7 \\ 2 & 4 & 11 & 2 & 10 \\ 7 & 14 & 12 & 5 & 8 \end{pmatrix}$$

5) a_1 =300 b_2 = 150
a_2 =250 b_3 = 120
a_3 =200 b_4 = 135
b_1 = 210 b_5 = 135

$$D = \begin{pmatrix} 4 & 8 & 13 & 2 & 7 \\ 9 & 4 & 11 & 9 & 17 \\ 3 & 16 & 10 & 1 & 4 \end{pmatrix}$$

6) a_1 =350 b_2 = 140
a_2 =200 b_3 = 200
a_3 =300 b_4 = 195
b_1 = 170 b_5 = 145

$$D = \begin{pmatrix} 22 & 14 & 6 & 28 & 30 \\ 19 & 17 & 26 & 36 & 36 \\ 3730 & 31 & 39 & 41 & \end{pmatrix}$$

7) a_1 =200 b_2 = 100
a_2 =250 b_3 = 120
a_3 =200 b_4 = 110
b_1 = 190 b_5 = 130

$$D = \begin{pmatrix} 28 & 27 & 18 & 27 & 24 \\ 18 & 26 & 27 & 32 & 21 \\ 27 & 33 & 23 & 31 & 34 \end{pmatrix}$$

8) a_1 =230 b_2 = 90
a_2 =250 b_3 = 160
a_3 =170 b_4 = 110
b_1 = 140 b_5 = 150

$$D = \begin{pmatrix} 40 & 19 & 25 & 25 & 35 \\ 49 & 26 & 27 & 18 & 38 \\ 46 & 27 & 36 & 40 & 45 \end{pmatrix}$$

9) a_1 =200 b_2 = 150
a_2 =300 b_3 = 120
a_3 =250 b_4 = 135
b_1 = 210 b_5 = 135

$$D = \begin{pmatrix} 20 & 10 & 13 & 13 & 18 \\ 27 & 19 & 20 & 16 & 22 \\ 36 & 17 & 19 & 21 & 23 \end{pmatrix}$$

10) a_1 =200 b_2 = 130
a_2 =350 b_3 = 190
a_3 =300 b_4 = 150
b_1 = 270 b_5 = 110

$$D = \begin{pmatrix} 24 & 50 & 55 & 27 & 16 \\ 50 & 47 & 23 & 17 & 21 \\ 35 & 59 & 55 & 27 & 41 \end{pmatrix}$$

11) a_1 =150 b_2 = 70
a_2 =150 b_3 = 130
a_3 =200 b_4 = 110
b_1 =100 b_5 = 90

$$D = \begin{pmatrix} 17 & 3 & 6 & 12 & 32 \\ 14 & 10 & 2 & 10 & 36 \\ 14 & 11 & 5 & 8 & 37 \end{pmatrix}$$

12) a_1 =330 b_2 = 170
a_2 =270 b_3 = 210
a_3 =350 b_4 = 150
b_1 =220 b_5 = 200

$$D = \begin{pmatrix} 10 & 12 & 24 & 50 & 42 \\ 13 & 22 & 49 & 66 & 32 \\ 26 & 27 & 35 & 67 & 63 \end{pmatrix}$$

13) a_1 =150 b_2 = 150
a_2 =200 b_3 = 75
a_3 =100 b_4 = 60
b_1 =90 b_5 = 75

$$D = \begin{pmatrix} 15 & 23 & 28 & 19 & 17 \\ 17 & 13 & 14 & 12 & 20 \\ 13 & 21 & 24 & 16 & 12 \end{pmatrix}$$

14) a_1 =300 b_2 = 195
a_2 =350 b_3 = 200
a_3 =200 b_4 = 140
b_1 =145 b_5 = 170

$$D = \begin{pmatrix} 18 & 31 & 35 & 25 & 13 \\ 16 & 25 & 21 & 9 & 9 \\ 45 & 30 & 25 & 33 & 41 \end{pmatrix}$$

15) a_1 =300 b_2 = 140
a_2 =300 b_3 = 115
a_3 =250 b_4 = 225
b_1 =150 b_5 = 220

$$D = \begin{pmatrix} 20 & 23 & 20 & 15 & 24 \\ 29 & 15 & 16 & 19 & 29 \\ 6 & 11 & 10 & 9 & 8 \end{pmatrix}$$

16) a_1 =300 b_2 = 150
a_2 =230 b_3 = 130
a_3 =320 b_4 = 180
b_1 =190 b_5 = 200

$$D = \begin{pmatrix} 25 & 20 & 22 & 31 & 32 \\ 11 & 19 & 18 & 18 & 20 \\ 26 & 30 & 17 & 19 & 20 \end{pmatrix}$$

17) a_1 =300 b_2 = 130
a_2 =250 b_3 = 150
a_3 =300 b_4 = 190
b_1 =130 b_5 = 250

$$D = \begin{pmatrix} 17 & 21 & 24 & 32 & 24 \\ 23 & 10 & 15 & 20 & 26 \\ 20 & 27 & 29 & 23 & 25 \end{pmatrix}$$

18) a_1 =200 b_2 = 140
a_2 =300 b_3 = 160
a_3 =250 b_4 = 180
b_1 =120 b_5 = 150

$$D = \begin{pmatrix} 16 & 25 & 26 & 26 & 23 \\ 25 & 30 & 30 & 32 & 33 \\ 34 & 25 & 23 & 26 & 32 \end{pmatrix}$$

19) a_1 =270 b_2 = 210

$$\begin{pmatrix} & & & & \\ & & & & \\ & & & & \end{pmatrix}$$

a_2 =450 b_3 = 200 37 30 15 19 37
a_3 =330 b_4 = 230 D = 16 19 13 19 21
b_1 =190 b_5 = 220 10 20 19 29 26

20) a_1 =210 b_2 = 220
a_2 =450 b_3 = 170 19 27 32 32 20
a_3 =290 b_4 = 210 D = 39 21 12 21 41
b_1 =200 b_5 = 150 15 14 28 27 20

21) a_1 =300 b_2 = 195
a_2 =350 b_3 = 200 10 12 24 50 42
a_3 =200 b_4 = 140 D = 13 22 49 66 32
b_1 =140 b_5 = 170 26 27 35 67 63

22) a_1 =210 b_2 = 220
a_2 =450 b_3 = 170 19 27 32 32 20
a_3 =290 b_4 = 210 D = 39 21 12 21 41
b_1 =200 b_5 = 150 15 14 28 27 20

23) a_1 =300 b_2 = 195
a_2 =350 b_3 = 200 10 12 24 50 42
a_3 =200 b_4 = 140 D = 13 22 49 66 32
b_1 =140 b_5 = 170 26 27 35 67 63

24) a_1 =200 b_2 = 125
a_2 =450 b_3 = 325 18 31 35 25 13
a_3 =250 b_4 = 250 D = 16 25 21 9 9
b_1 =100 b_5 = 100 45 30 25 33 41

25) a_1 =200 b_2 = 100
a_2 =250 b_3 = 120 18 27 28 24 27
a_3 =200 b_4 = 110 D = 27 26 18 21 32
b_1 = 190 b_5 = 130 23 33 27 34 31

26) a_1 =150 b_2 = 150
a_2 =200 b_3 = 75 13 21 24 16 12
a_3 =100 b_4 = 60 D = 17 13 14 12 20
b_1 =90 b_5 = 75 15 23 28 19 17

27) a_1 =200 b_2 = 100
a_2 =250 b_3 = 120 5 7 4 2 5
a_3 =200 b_4 = 110 D = 7 1 3 1 10
b_1 =190 b_5 = 130 2 3 6 8 7

28) $a_1 = 270$ $b_2 = 210$
$a_2 = 450$ $b_3 = 200$
$a_3 = 330$ $b_4 = 230$
$b_1 = 190$ $b_5 = 220$

$$D = \begin{pmatrix} 7 & 3 & 10 & 1 & 3 \\ 6 & 1 & 11 & 9 & 1 \\ 1 & 2 & 19 & 19 & 6 \end{pmatrix}$$

29) $a_1 = 200$ $b_2 = 130$
$a_2 = 350$ $b_3 = 190$
$a_3 = 300$ $b_4 = 150$
$b_1 = 270$ $b_5 = 110$

$$D = \begin{pmatrix} 22 & 14 & 16 & 28 & 30 \\ 19 & 17 & 26 & 36 & 36 \\ 37 & 30 & 31 & 39 & 41 \end{pmatrix}$$

30) $a_1 = 200$ $b_2 = 130$
$a_2 = 175$ $b_3 = 80$
$a_3 = 225$ $b_4 = 190$
$b_1 = 100$ $b_5 = 100$

$$D = \begin{pmatrix} 15 & 17 & 24 & 22 & 25 \\ 17 & 11 & 13 & 21 & 30 \\ 12 & 13 & 16 & 18 & 17 \end{pmatrix}$$

6.4 Лабораторная работа №4. Решение задачи целочисленного программирования

1. Записать задачу целочисленного программирования.
2. Решить задачу методом Гомори.

Задача целочисленного программирования приводится к каноническому виду.

Модуль Gomori;

Входные данные:

- m – число ограничений;
- n – число переменных;
- C – вектор коэффициентов целевой функции;
- A – матрица коэффициентов системы ограничений и правых частей (ввод построчный);

Выходные данные:

- x – вектор целочисленныых значений оптимального решения;
- z –оптимальное значение целевой функции.

Пример.

$$f = 5x_1 + 4x_2 \rightarrow \max$$

$$\begin{cases} 1x_1 + 1x_2 + 1x_3 = 18 \\ 5x_1 - 1x_2 + 1x_4 = 20 \\ 1x_1 - 1x_2 + 1x_5 = 8 \end{cases}$$

Текст программы.

```
Program Gomori;
uses crt;
var
  f1,f2: text;
  a : array[1..100,0..100] of real;
  c,b : array[1..100] of real;
  d : array[0..100] of real;
  k : array[0..100] of byte;
  kt,kt2,dl: integer;
  i,j,m,n,mi,mj,r,x,y : byte;
  dj,min,max : real;
  s : string[12];
  st,s1,s2,s3 : string[10];
```

```
  ch : char;

{ВЫВОД В ФАЙЛ ЧИСЛА В ФОРМАТЕ}
procedure wr(r:real;b:boolean);
var w: byte;
begin
if (abs(frac(r)-round(frac(r)))>0.0001) or (r>1.0e10) then str(r:4:2,st)
   else str(round(r),st);
if b then for w:=length(st) to 10 do st:=st+' ';
write(f2,st);
end;

{ВЫВОД В ФАЙЛ ТАБЛИЦ}
procedure writetablefile;
var ws:string[10];
begin
writeln(f2,#10,#13,' ИТЕРАЦИЯ_____',kt2,'.',kt);
if s='' then dl:=79 else dl:=n*12;
for i:=1 to dl do write(f2,'=');
write(f2,#10,#13,'bx  ');
for i:=0 to n do
    begin
    if (s='') and (i>6) then break;
    if (abs(frac(c[i])-round(frac(c[i])))>0.2) or (c[i]>1.0e10)
       then str(c[i]:4:2,st)  else str(round(c[i]),st);
    str(i,ws);
    st:='a'+ ws + '=' + st;
    for j:=length(st) to 11 do st:=st+' ';
    write(f2,st);
    end;
writeln(f2);
for i:=1 to dl do write(f2,'-');
for i:=1 to m do
    begin
    write(f2,#10, #13, 'x', k[i], '  ');
    if (s='') and (i>15) then break;
    for j:=0 to n do
        begin
        if (s='') and (j>6) then break;
        wr(a[i,j],true);
        end;
    end;
writeln(f2);
for i:=1 to dl do write(f2,'-');
write(f2,#10,#13,'z:  ');
```

```
for j:=0 to n do
    begin
    if (s='') and (j>6) then break;
    wr(d[j],true);
    end;
writeln(f2);
for i:=1 to dl do write(f2,'=');
end;

{ПЕРЕСЧЕТ ПО ПРАВИЛУ ПРЯМОУГОЛЬНИКА (ЖОРДАН-ГАУСС)}
procedure gauss;
begin
for i:=1 to m do
   for j:=0 to n do
      if (i<>mi) and (j<>mj) then
         a[i,j]:=a[i,j]-a[mi,j]*a[i,mj]/a[mi,mj];
for i:=1 to m do if i<>mi then a[i,mj]:=0;
d[mj]:=0;
for j:=0 to n do a[mi,j]:=a[mi,j]/min;
end;

{СИМПЛЕКС МЕТОД}
procedure simplex;
begin
for i:=1 to m do
  begin
{ЗАНОСИТСЯ НОМЕР БАЗИСНОЙ ПЕРЕМЕННОЙ И ЕЕ НОМЕР}
   for j:=n downto 1 do if a[i,j]<>0 then
     begin
     b[i]:=c[j]; k[i]:=j; break;
     end;
  end;

r:=0;
{ПЕРЕСЧЕТ z}
repeat
for j:=0 to n do
   begin
   dj:=0;
   for i:=1 to m do dj:=dj+b[i]*a[i,j];
   d[j]:=dj-c[j];
   end;
writetablefile;
inc(kt);
min:=0;mj:=0;
```

```
{ВЫБОР НАПРАВЛЯЮЩЕГО СТОЛБЦА}
for j:=1 to n do
   if min>d[j] then begin min:=d[j]; mj:=j; end;
if mj=0 then break;
mi:=0; min:=1.7e38;
{ВЫБОР НАПРАВЛЯЮЩЕЙ СТРОКИ}
for i:=1 to m do
   if (a[i,mj]>0) and (min>a[i,0]/a[i,mj]) then
      begin
      min:=a[i,0]/a[i,mj];
      mi:=i;
      end;
if mi=0 then begin r:=2; break; end;
min:=a[mi,mj];
gauss;
b[mi]:=c[mj];
writeln(f2,#13,#10,'  x',k[mi],'=>x',mj);
k[mi]:=mj;
if s2='con' then if s='' then ch:=readkey;
until ch=#27;
end;

{ДВОЙСТВЕННЫЙ СИМПЛЕКС МЕТОД}
procedure doublesimplex;
begin
r:=0;
repeat
writetablefile;
inc(kt);
min:=0;
{ВЫБОР НАПРАВЛЯЮЩЕЙ СТРОКИ}
for i:=1 to m do if min>a[i,0] then begin min:=a[i,0]; mi:=i; end;
if min=0 then break;
min:=1.7e38;
{ВЫБОР НАПРАВЛЯЮЩЕГО СТОЛБЦА}
for j:=1 to n do if a[mi,j]<-0.001 then
   begin
   dj:=-d[j]/a[mi,j];
   if min>dj then begin min:=dj; mj:=j; end;
   end;
if min=1.7e38 then begin r:=2; writetablefile; break; end;
min:=a[mi,mj];
b[mi]:=c[mj];
writeln(f2,#10,#13,'  x',k[mi],'=>x',mj);
k[mi]:=mj;
```

```
gauss;
for j:=0 to n do
   begin
   dj:=0;
   for i:=1 to m do begin
   dj:=dj+b[i]*a[i,j];
   delay(1);  end;
   d[j]:=dj-c[j];
   end;
if s2='con' then if s='' then ch:=readkey;
until ch=#27;
end;

{ВЫВОД РЕЗУЛЬТАТА X=( ) }
procedure result;
var res:byte;
begin
if r=2 then write(f2,#13,#10,'Нет решения !')
 else begin
      write(f2,#13,#10,' x=( ');
      for i:=1 to n do
         begin
         res:=0;
         for j:=1 to m do if k[j]=i then res:=j;
         if res<>0 then begin wr(a[res,0],false); write(f2,'; '); end
            else write(f2,' 0;');
         end;
      writeln(f2,')');
      write(f2,' z= '); wr(d[0],false);
      end;
         writeln(f2);
         writeln(f2,'      ');
         write(f2,'     .B.');

end;

procedure klv;
begin
    writeln('Введите число ограничений и число переменных+число
ограничений');
    read(m); readln(n);
    writeln('Задача вводится в канонической форме !');
    writeln('Введите заданные коэффициенты целевой функции');
    for i:=1 to n do read(c[i]);
    writeln('Введите известные коэффициенты в системе ограничений');
```

```
    writeln('      и произвольные коэффициенты по-строчно');
    for i:=1 to m do begin
                      for j:=1 to n do read(a[i,j]);
                      read(a[i,0]);
                    end; readln;
end;

procedure fil;
begin
    writeln('Введите имя исходного файла');
    readln(s1);
    {ОТКРЫТИЕ ИСХОДНОГО ФАЙЛА }
    {$i-}
    if s1='' then
    begin
        writeln('Ошибка открытия файла !');
        writeln('Не задано имя !')
    end
    else
    begin
        assign(f1,s1);
        reset(f1);
        read(f1,m); readln(f1,n);
        for i:=1 to n do read(f1,c[i]);
        for i:=1 to m do begin
                           for j:=1 to n do read(f1,a[i,j]);
                           read(f1,a[i,0]);
                         end;
        close(f1);
    end;
end;

{=======================================================}

begin
    clrscr;
    writeln('Ввод исходных данных "0"-с клавиатуры, "1"-из файла');
    readln(s3);
    if s3='0'then klv;
    if s3='1' then fil;

    writeln('Введите имя файла для записи результата или "con" для вывода
на экран');
    readln(s2);
```

```
    if s2='' then writeln('НЕ ЗАДАНО УСТРОЙСТВО ВЫВОДА !')
    else
    begin
        assign(f2,s2);
        rewrite(f2);
        clrscr;
        {ПРОВЕРКА НА ПРАВИЛЬНУЮ КАНОНИЧЕСКУЮ ФОРМУ}
        for i:=1 to m do if a[i,0]<0 then
           begin
              write('НЕ ПРАВИЛЬНАЯ КАНОНИЧЕСКАЯ ФОРМА !');
              readkey;
              exit;
           end;

        {ВЫВОД В ФАЙЛ ЗАДАЧИ}
        writeln(f2,'РАЗМЕРНОСТЬ ЗАДАЧИ - ',m,'x',n);
        write(f2,'   ');
        for i:=1 to n do
        begin
            if (c[i]>0) and (i=1) then begin wr(c[i],false); write(f2,'x',i); end;
            if (c[i]>0) and (i>1) then begin write(f2,'+'); wr(c[i],false);
write(f2,'x',i); end;
            if c[i]<0 then begin wr(c[i],false); write(f2,'x',i); end;
        end;
        writeln(f2,' =>max');
        for i:=1 to m do
        begin
            write(f2,i,') ');
            for j:=1 to n do
            begin
                if (a[i,j]>0) and (j=1) then begin wr(a[i,j],false); write(f2,'x',j); end;
                if (a[i,j]>0) and (j>1) then begin write(f2,'+'); wr(a[i,j],false);
write(f2,'x',j); end;
                if a[i,j]<0 then begin wr(a[i,j],false); write(f2,'x',j); end;
            end;
            write(f2,'='); wr(a[i,0],false); writeln(f2);
        end;
        if s2='con' then if s='' then ch:=readkey;
        if ch=#27 then exit;
        kt:=0; kt2:=0;
        simplex;
        if ch=#27 then exit;
        result;
        if r=2 then
        begin
```

```
            if s2='con' then  if s='' then readkey;
            close(f2);
            exit;
        end;
        if s2='con' then if s='' then ch:=readkey;
        if ch=#27 then exit;
        {МЕТОД ГОМОРИ}
        repeat
            r:=0;
            inc(kt2);
            for i:=1 to m do if abs(a[i,0]-round(a[i,0]))>0.0001 then begin r:=1;
break; end;
            if r=0 then
            begin
                close(f2);
                exit;
            end
            else writeln(f2,#10,#13,#10,' ПЕРЕМЕННЫЕ НЕЦЕЛОЧИСЛЕН-
НЫЕ : ФОРМИРУЕМ ОТСЕЧЕНИЕ.');
            if (n+2)<102 then begin inc(n); inc(m); end
            else begin
                    write(f2, #13, #10, 'ОШИБКА : ПЕРЕПОЛНЕНИЕ !');
                    if s2='con' then if s='' then readkey;
                    close(f2);
                    exit;
            end;
            k[m]:=n;
            min:=50;
            for i:=1 to m-1 do if (abs(a[i,0]-round(a[i,0]))>0.0001)  and (min>k[i])
then
            begin
                mi:=i;
                min:=k[i];
            end;
            for j:=0 to n-1 do if a[mi,j]>=0 then a[m,j]:=-frac(a[mi,j])
               else a[m,j]:=-(a[mi,j]+abs(int(a[mi,j]))+1);
            a[m,n]:=1;
            kt:=0;
            doublesimplex;
            if ch=#27 then exit;
            result;
            if s2='con' then if s='' then ch:=readkey;
        until ch=#27;
        close(f2);
    end;
```

end.

Входные данные.

m=3

n=5

Вектор коэффициентов целевой функции

5 4 0 0 0

Матрица коэффициентов системы ограничений и ее правых частей построчно

1 1 1 0 0 18

5 -1 0 1 1 20

1 -1 0 0 1 8

Результаты расчета.

$\boldsymbol{x}$=(6;12;0;2;14;0)

z=78.

Варианты заданий.

1 $f(\boldsymbol{x})=x_1+2x_2 \rightarrow \max(\min)$

$$\begin{cases} 2x_1 - x_2 \geq 6 \\ 2x_1 + x_2 \leq 1 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

2 $f(\boldsymbol{x})=3x_1+2x_2 \rightarrow \max(\min)$

$$\begin{cases} 2x_1 + x_2 \geq 12 \\ x_1 + x_2 \leq 1 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

3 $f(\boldsymbol{x})=x_1+3x_2 \rightarrow \max(\min)$

$$\begin{cases} -x_1 + x_2 \geq 6 \\ x_1 - 2x_2 \geq 10 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

4 $f(\boldsymbol{x})=3x_1+x_2 \rightarrow \max(\min)$

$$\begin{cases} -x_1 - 2x_2 \geq 8 \\ -2x_1 + x_2 \leq 2 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

5 $f(\boldsymbol{x})=x_1+x_2 \rightarrow \max(\min)$

$$\begin{cases} x_1 - 2x_2 \geq 14 \\ -5x_1 + 3x_2 \geq 15 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

6 $f(\boldsymbol{x})=-2x_1-3x_2 \rightarrow \max(\min)$

$$\begin{cases} -4x_1 + 2x_2 \geq 4 \\ x_1 - x_2 \geq 6 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

7 $f(\boldsymbol{x})=5x_1+7x_2 \rightarrow \max(\min)$

$$\begin{cases} 5x_1 - 6x_2 \leq 30 \\ x_1 - 4x_2 \leq 28 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

8 $f(\boldsymbol{x})=x_1+x_2 \rightarrow \max(\min)$

$$\begin{cases} -2x_1 - x_2 \geq 6 \\ -x_1 + 3x_2 \leq 9 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0$$

9 $f(\boldsymbol{x})=x_1+3x_2\to\max\ (\min)$

$$\begin{cases}-7x_1+4x_2\ge 28\\ x_1-3x_2\ge 15\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

10 $f(\boldsymbol{x})=x_1+2x_2\to\max\ (\min)$

$$\begin{cases}x_1-6x_2\ge 6\\ -2x_1+x_2\ge 2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

11 $f(\boldsymbol{x})=x_1+3x_2\to\max\ (\min)$

$$\begin{cases}3x_1+4x_2\le 12\\ 2x_1-x_2\ge 10\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

12 $f(\boldsymbol{x})=x_1+5x_2\to\max\ (\min)$

$$\begin{cases}-x_1-x_2\ge 12\\ x_1-4x_2\le 8\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

13 $f(\boldsymbol{x})=2x_1+3x_2\to\max(\min)$

$$\begin{cases}x_1-4x_2\ge 12\\ -4x_1+x_2\ge 4\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

14 $f(\boldsymbol{x})=2x_1+3x_2\to\max\ (\min)$

$$\begin{cases}x_1+x_2\ge 10\\ 2x_1+x_2\le 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

15 $f(\boldsymbol{x})=3x_1+x_2\to\max\ (\min)$

$$\begin{cases}-7x_1+3x_2\ge 21\\ x_1-5x_2\ge 10\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

16 $f(\boldsymbol{x})=3x_1+2x_2\to\max(\min)$

$$\begin{cases}4x_1+3x_2\le 8\\ x_1+x_2\ge 4\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

17 $f(\boldsymbol{x})=3x_1+x_2\to\max\ (\min)$

$$\begin{cases}x_1-2x_2\ge 6\\ x_1+5x_2\ge 8\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

18 $f(\boldsymbol{x})=x_1+4x_2\to\max\ (\min)$

$$\begin{cases}2x_1-2x_2\ge 14\\ -x_1-x_2\ge 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

19 $f(\boldsymbol{x})=2x_1+x_2\to\max(\min)$

$$\begin{cases}2x_1+x_2\ge 10\\ x_1-x_2\ge 8\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

20 $f(\boldsymbol{x})=x_1+3x_2\to\max\ (\min)$

$$\begin{cases}x_1-2x_2\ge 6\\ -x_1+3x_2\ge 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

21 $f(\boldsymbol{x})=5x_1+4x_2\to\max(\min)$

$$\begin{cases}x_1-5x_2\ge 20\\ -x_1+x_2\ge 9\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

22 $f(\boldsymbol{x})=x_1-x_2\to\max\ (\min)$

$$\begin{cases}2x_1-3x_2\ge 12\\ x_1+x_2\le 1\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

23 $f(\boldsymbol{x})=6x_1+2x_2 \to \max\ (\min)$

$$\begin{cases} -3x_1+x_2 \ge 6 \\ -3x_1+x_2 \le 1 \\ x_1+4x_2 \le 28 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

24 $f(\boldsymbol{x})=2x_1+x_2 \to \max\ (\min)$

$$\begin{cases} -x_1+x_2 \ge 1 \\ x_1-2x_2 \ge 1 \\ x_1-x_2 \le 8 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

25 $f(\boldsymbol{x})=x_1+x_2 \to \max\ (\min)$

$$\begin{cases} 3x_1-4x_2 \ge 12 \\ x_1+x_2 \le 1 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

26 $f(\boldsymbol{x})=2x_1+x_2 \to \max\ (\min)$

$$\begin{cases} x_1-5x_2 \ge 10 \\ x_1-2x_2 \le 6 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

27 $f(\boldsymbol{x})=6x_1+x_2 \to \max\ (\min)$

$$\begin{cases} -x_1+x_2 \ge 15 \\ x_1-3x_2 \ge 9 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

29 $f(\boldsymbol{x})=4x_1+x_2 \to \max\ (\min)$

$$\begin{cases} 6x_1+x_2 \ge 12 \\ x_1+2x_2 \le 10 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

28 $f(\boldsymbol{x})=2x_1+x_2 \to \max\ (\min)$

$$\begin{cases} x_1+x_2 \ge 2 \\ x_1+2x_2 \ge 8 \\ 3x_1-x_2 \le 6 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

30 $f(\boldsymbol{x})=5x_1+2x_2 \to \max\ (\min)$

$$\begin{cases} x_1-3x_2 \ge 9 \\ 3x_1-x_2 \le 6 \\ 2x_2 \le 12 \end{cases}$$

$$x_1 \ge 0, x_2 \ge 0$$

6.5 Лабораторная работа №5. Решение задачи о коммивояжере

1. Решение задачи методом ветвей и границ.

Модуль Traveller

Входные данные

Программа позволяет вводить исходные данные в двух форматах:

- готовую матрицу расстояний;
- набор координат точек, через которые проходит путь коммивояжера.

В последнем случае матрица расстояний высчитывается автоматически.

Ввести входные данные можно:

- с клавиатуры;
- из указанного текстового файла.

Координаты точек в файле должны располагаться в двух строках: в первой – координаты каждой точки по оси X, во второй – по оси Y.

Матрица расстояний задается построчно.

При вводе данных из файла, число точек находится автоматически. При вводе же с клавиатуры оно задается.

Выходные данные.

Все основные выходные данные, включая матрицы расстояний для каждой итерации, оценки ветвления и т.п., выводятся в файл output.res в текущем каталоге.

При наличии в текущем каталоге видеодрайвера egavga.bgi программа также покажет результат графически (при задании координат точек).

Пример.

Имеется четыре пункта, расстояние между которыми описано матрицей расстояний. Найти оптимальный (минимальный) замкнутый маршрут объезда городов.

$$\begin{pmatrix} \infty & 13 & 12 & 4 \\ 13 & \infty & 7 & 8 \\ 12 & 7 & \infty & 5 \\ 4 & 8 & 5 & \infty \end{pmatrix}$$

Текст программы.

```
Program Traveller;
Uses Crt,Graph;
const
 N=30;
 CurrentN:word=N;
 BinMapSize:word=N;
 NoWay=-1;
type
 TVector=array [0..N] of single;
 TMap=array [0..N] of TVector;
 TPoint=record  X:single;
  Y:single;
 end;
 TPointVector=array [1..N] of TPoint;
var
 Map:TMap;
 BinMap:TMap;
 Points:TPointVector;
Procedure LoadPointsFromFile(FileName:string);
var
 f:text;
 i:word;
begin
 assign(f,FileName);
 {$I-}
 reset(f);
 {$I+}
 if IOResult<>0 then
  begin
   writeln(  ,FileName,');
   halt(1);
  end;
 i:=1;
 while not EoLn(f) do
  begin
   read(f,Points[i].X);
   inc(i);
  end;
 CurrentN:=i-1;
 for i:=1 to CurrentN do
  read(f,Points[i].Y);
 close(f);
end;
```

```
Procedure LoadPointsFromCon;
var i:word;
begin
 readln(CurrentN);
 for i:=1 to CurrentN do
  begin
   write('Point N',i,' X=');
   readln(Points[i].x);
   write('Point N',i,' Y=');
   readln(Points[i].y);
  end;
end;
Procedure ClearBitMap;
var i,j:word;
begin
 for i:=1 to CurrentN do
  for j:=1 to CurrentN do
   BinMap[i][j]:=0;
 BinMapSize:=CurrentN;
end;
Procedure PointsToMap;
var i,j:word;
begin
 for i:=1 to CurrentN do
  begin
   Map[0][i]:=i;
   Map[i][0]:=i;
  end;
 for i:=1 to CurrentN do
  begin
   for j:=1 to CurrentN do
   if i<>j then
    begin
     Map[i][j]:=SQRT(SQR(abs(Points[i].X-Points[j].X))+SQR(abs(Points[i].Y-
Points[j].Y)));
    end else Map[i][j]:=NoWay;
  end;
end;
Procedure LoadMapFromFile(FileName:string);
var f:text;
    i,j:word;
    k:single;
begin
 assign(f,FileName);
 {$I-}
```

```
reset(f);
 {$I+}
 if IOResult<>0 then
  begin
   writeln(' "',FileName,'".');
   halt(1);
  end;
 i:=1;
 while not EOF(f) do
  begin
   Map[0][i]:=i;
   Map[i][0]:=i;
   j:=1;
   while not EOLN(f) do
    begin
     read(f,k);
     if i=j then k:=NoWay;
     Map[i][j]:=k;
     inc(j);
    end;
   inc(i);
   readln(f);
  end;
 CurrentN:=i-1;
 close(f);
end;
Function GetMarkIJ(M:TMap;i,j:word):single;

var i1,j1:word;
    tmp:TVector;
    k1,k2:single;
begin
 M[i][j]:=NoWay;
 k1:=0;
 k2:=0;
 for i1:=1 to CurrentN do
  if M[i][i1]<>NoWay then
   begin
    k1:=M[i][i1];
    break;
   end;
 for i1:=1 to CurrentN do
  if M[i1][j]<>NoWay then
   begin
    k2:=M[i1][j];
```

```
    break;
   end;
 for i1:=1 to CurrentN do
  if (M[i][i1]<k1)and(M[i][i1]<>NoWay) then
   k1:=M[i][i1];
 for i1:=1 to CurrentN do
  if (M[i1][j]<k2)and(M[i1][j]<>NoWay) then
   k2:=M[i1][j];
 GetMarkIJ:=k1+k2;
end;
Procedure GetHeaviestZero(M:TMap;var i,j:word;var q:single;CurrentN:word);
var
 i1,j1:word;
 max,m1:single;
 t:boolean;
begin
 max:=0;
 i:=0;J:=0;
 q:=0;
 t:=true;
 for i1:=1 to CurrentN do
  begin
   for j1:=1 to CurrentN do
   if M[i1][j1]=0 then
    begin
     m1:=GetMarkIJ(M,i1,j1);
     if t then
      begin
       t:=false;
       max:=m1;
       i:=i1;
       j:=j1;
      end;
     if m1>max then
      begin
       max:=m1;
       i:=i1;
       j:=j1;
      end;
    end;
  end;
  q:=max;
end;
Procedure ReduceMap(var M:TMap;CurrentN:word;var res:single);
var
```

```
    i,j:word;
    colm,rowm:single;
begin
 res:=0;
 for i:=1 to CurrentN do
  begin
    for j:=1 to CurrentN do
     if M[i][j]<>NoWay then
      begin
       rowm:=M[i][j];
       break;
      end;
    for j:=1 to CurrentN do
     if M[i][j]<>NoWay then
      begin
       if M[i][j]<rowm then rowm:=M[i][j];
      end;
    if rowm>0 then
    begin
     for j:=1 to CurrentN do
      if Map[i][j]<>NoWay then
        Map[i][j]:=Map[i][j]-rowm;
     res:=res+rowm;
    end;
  end;
 for j:=1 to CurrentN do
  begin
    colm:=0;
    for i:=1 to CurrentN do
     if Map[i,j]<>NoWay then
      begin
       colm:=M[i][j];
       break;
      end;
    for i:=1 to CurrentN do
     if M[i][j]<>NoWay then
      begin
       if M[i][j]<colm then colm:=M[i][j];
      end;
    if colm>0 then
    begin
     for i:=1 to CurrentN do
      if Map[i,j]<>NoWay then
       Map[i,j]:=Map[i,j]-colm;
     res:=res+colm;
```

```
    end;
  end;
end;
Procedure PrintMap(M:TMap;toFile:byte;var f:text);

var i,j:word;
begin
 for i:=0 to CurrentN do
  begin
   for j:=0 to CurrentN do
    begin
     if M[i][j]<>NoWay then
     begin
     case toFile of
     0:write(' ',M[i][j]:5:1);
     1:write(f,' ',M[i][j]:5:1);
     2:begin write(f,' ',M[i][j]:5:1); write(' ',M[i][j]:5:1); end;
     end;
     end else
     begin
     case toFile of
     0:write('*':6);
     1:write(f,'*':6);
     2:begin write(f,'*':6); write('*':6); end;
     end;
     end;
    end;
     case toFile of
     0:writeln;
     1:writeln(f);
     2:begin writeln(f); writeln; end;
     end;
  end;
end;
Procedure ShowPoints(k:single);

var i:word;
    s:string;
begin
 for i:=1 to CurrentN do
  begin
   setcolor(Yellow);
   circle(round(Points[i].X*k),round(Points[i].Y*k),2);
   str(i,s);
   setcolor(red);
```

```
   outtextxy(round(Points[i].X*k)+2,round(Points[i].Y*k)+2,s);
  end;
end;
var i,j:word;
    out:text;
    tmp:TVector;
Procedure PointToPointWay(M:TMap;var V:TVector);
var ind,j:word;
Procedure NextPoint(z:word;point:word);
var i:word;
begin
 for i:=1 to BinMapSize do
  if (M[point][i]=1)and(i<>z) then
   begin
    V[ind]:=i;
    inc(ind);
    NextPoint(point,i);
   end;
end;
begin
  ind:=1;
  NextPoint(0,1);
  write('  1  - ');
  write(out,'  1  - ');
  for j:=1 to ind-1 do
   begin
    write(V[j]:1:0,'  - ');
    write(OUT,V[j]:1:0,'  - ');
   end;
  writeln(1:3);
  writeln(OUT,1:3);
end;
Function AreConnected(p1,p2:word;B:TMap;CurrentN:word):boolean;
var l:word;
Procedure Next(predp,p:word);
var i:word;
begin
 for i:=1 to CurrentN do
 if (B[p][i]=1)and(i<>predp) then
  if i<>p2 then
  begin
   Next(p,i);
   Break;
  end
   else
```

```
    begin
     l:=p2;
     Exit;
    end;
end;
begin
 l:=0;
 Next(0,p1);
 if l=p2 then
  AreConnected:=true else
  AreConnected:=false;
end;
Procedure SetToInfinity(var M:TMap;i,j:word);
var i1,j1:word;
    t:boolean;
begin
 t:=true;
 for j1:=1 to CurrentN do
  if M[0][j1]>=j then break;
 if M[0][j1]<>j then t:=false;
 if t then
  begin
   for i1:=1 to CurrentN do
    if M[i1][0]>=i then break;
   if M[i1][0]<>i then t:=false;
  end;
 if t then M[i1][j1]:=NoWay;
end;
Procedure ExcludeWays(var Map:TMap);
var i,j:word;
begin
 for i:=1 to BinMapSize-1 do
  for j:=i+1 to BinMapSize do
   begin
    if AreConnected(i,j,BinMap,BinMapSize) then
     begin
      SetToInfinity(Map,i,j);
      SetToInfinity(Map,j,i);
     end;
    end;
end;
Function GetPointNumX(n:word):word;

begin
 GetPointNumX:=round(Map[n][0]);
```

```
end;
Function GetPointNumY(n:word):word;
begin
 GetPointNumY:=round(Map[0][n]);
end;
Procedure CutMatrix(var M:TMap;i,j:word;var CurrentN:word);
var ie,je,s:single;
    i1,j1:word;
    tmp1:TMap;
begin
 tmp1:=M;
 ie:=M[i][0];
 je:=M[0][j];
 ExcludeWays(M);
 for i1:=0 to CurrentN do
  for j1:=j to CurrentN-1 do M[i1][j1]:=M[i1][j1+1];
 for j1:=0 to CurrentN do
  for i1:=i to CurrentN-1 do M[i1][j1]:=M[i1+1][j1];
 Dec(CurrentN);
end;
var tmp1,tmp2:TMap;
    m1x,m2x:word;
    m1y,m2y:word;
    s1,q1:single;
    NodeCost,NonCutted,Cutted:single;
    k:char;
    NoVisiblePoints:boolean;
    fn:string;
label 1;
Procedure InitGraphicMode(path:string);
var d,m:integer;
begin
 d:=detect;
 Initgraph(d,m,path);
 if GraphResult<>0 then
  begin
   ClrScr;
   writeln('');
   writeln('“');
   NoVisiblePoints:=true;
  end;
end;
Procedure CloseGraphicMode;
begin
 CloseGraph;
```

```
end;
BEGIN
 assign(out,'output.res');
 rewrite(out);
 ClrScr;
 writeln('m');
 repeat
  k:=readkey;
  case k of
  '1':
   begin
    writeln('‡Ђѓђ“‡ЉЂ ’Ћ—…Љ €‡ ”Ђ‰‹Ђ');
    write(' ');
    readln(fn);
    LoadPointsFromFile(fn);
    PointsToMap;
    NoVisiblePoints:=false;
   end;
  '2':
   begin
    writeln('‡Ђѓђ“‡ЉЂ ЊЂ’ђ€–› ђЂ‘‘’ЋµЌ€‰ €‡ ”Ђ‰‹Ђ');
    write(' ');
    readln(fn);
    LoadMapFromFile(fn);
    NoVisiblePoints:=true;
   end;
  '3':
   begin
    writeln(',,Ћ,, ’Ћ—…Љ');
    write(' ');
    LoadPointsFromCon;
    PointsToMap;
    NoVisiblePoints:=false;
   end;
{  '4':
   begin
    writeln('TYPING WAYS-MATRIX');
    write('Enter points count: ');
   end; }
  '5':
   begin
    Halt(0);
   end;
  end;
 until (k in ['1','2','3','5']);
```

```
 ClearBitMap;
 if not NoVisiblePoints then
 InitGraphicMode('');
 if not NoVisiblePoints then
 ShowPoints(2);
 if not NoVisiblePoints then
 readkey;
 writeln(out,' ');
 writeln(out,'¬Ґв®¤®¬ ўЃвўЃ© Ё Ja -Ёж.');
 writeln(out);
 writeln(out,'');
 writeln(out);
 PrintMap(Map,1,out);
 writeln(out);
 ReduceMap(Map,CurrentN,NodeCost);
1:
 PrintMap(Map,1,out);
 writeln(out);

 GetHeaviestZero(Map,i,j,q1,CurrentN);
 NonCutted:=NodeCost+q1;
 tmp1:=Map;
 tmp1[i][j]:=NoWay;
 tmp2:=Map;
 m1x:=GetPointNumX(i);
 m1y:=GetPointNumY(j);
 BinMap[m1x,m1y]:=1;
 BinMap[m1y,m1x]:=1;
 CutMatrix(tmp2,i,j,CurrentN);
 ReduceMap(tmp2,CurrentN,s1);
 Cutted:=NodeCost+s1;
 writeln(out,'‘',GetPointNumY(j),'.');
 writeln(out,' writeln(out,'----------------------------');
 writeln(out,'ЋжЃ-Є  г§«      = ',NodeCost:5:6);
 writeln(out,'ЋжЃ-Є  {',m1x,',',m1y,  '}    = ',Cutted:5:6);
 writeln(out,'ЋжЃ-Є  -Ѓ {',m1x,',',m1y,'} = ',NonCutted:5:6);
 writeln(out,'----------------------------');
 if (NonCutted<Cutted)and(CurrentN>1) then
  begin
   Inc(CurrentN);
   writeln(out,'',GetPointNumY(j),'] ¬Ґ-миЃ, ',#13#10,
            ['',GetPointNumX(i),'-',GetPointNumY(j),'] ...');
   Map:=tmp1;
   BinMap[GetPointNumX(i),GetPointNumY(j)]:=0;
   BinMap[GetPointNumY(j),GetPointNumX(i)]:=0;
```

```
   NodeCost:=NonCutted;
  end else
 if (NonCutted>Cutted) then
  begin
   writeln(out,'',GetPointNumY(j),'] ¬Ґ-миҐ, ',#13#10,
 ['',GetPointNumX(i),'-',GetPointNumY(j),'] ...');
   writeln(out,' ',GetPointNumY(j),'.');
   if not NoVisiblePoints then

line(round(Points[GetPointNumX(i)].x*2),round(Points[GetPointNumX(i)].y*2)
,

round(Points[GetPointNumY(j)].x*2),round(Points[GetPointNumY(j)].y*2));
   Map:=tmp2;
   NodeCost:=Cutted;
  end else
  begin
   writeln(out,' [',GetPointNumX(i),'-',GetPointNumY(j),'] ',#13#10,
            ['',GetPointNumX(i),'-',GetPointNumY(j),']
 ['',GetPointNumX(i),'-',GetPointNumY(j),'] ...');
   writeln(out,' ',GetPointNumY(j),'.');
   if not NoVisiblePoints then

line(round(Points[GetPointNumX(i)].x*2),round(Points[GetPointNumX(i)].y*2)
,

round(Points[GetPointNumY(j)].x*2),round(Points[GetPointNumY(j)].y*2));
   Map:=tmp2;
   NodeCost:=Cutted;
  end;

 writeln(out);
 ReduceMap(Map,CurrentN,s1);
 if CurrentN>1 then goto 1;
  writeln(out,'',GetPointNumY(1),']. ,лЎЁа Ґ¬ ҐJ®.');
  writeln(out);
  writeln(out,'');
   if not NoVisiblePoints then

line(round(Points[GetPointNumX(1)].x*2),round(Points[GetPointNumX(1)].y*
2),

round(Points[GetPointNumY(1)].x*2),round(Points[GetPointNumY(1)].y*2));
 if not NoVisiblePoints then
 readkey;
```

```
 if not NoVisiblePoints then
 CloseGraphicMode;
 writeln(out,'');
 PointToPointWay(BinMap,tmp);
 Writeln('‘"output.res" ');
close(out);
END.
```

Входные данные из файла Z1

```
0  13 12 4
13 0  7  8
12 7  0  5
4  8   5  0
```

Результаты расчета ,записанные в файл OUTPUT.REZ
Решение задачи коммивояжера
методом ветвей и границ.

Исходная матрица расстояний:

```
   0.0   1.0   2.0   3.0   4.0
   1.0     *  13.0  12.0   4.0
   2.0  13.0     *   7.0   8.0
   3.0  12.0   7.0     *   5.0
   4.0   4.0   8.0   5.0     *

   0.0   1.0   2.0   3.0   4.0
   1.0     *   7.0   8.0   0.0
   2.0   6.0     *   0.0   1.0
   3.0   7.0   0.0     *   0.0
   4.0   0.0   2.0   1.0     *
```

Самый "тяжелый" нуль получен в строке 1, столбце 4.
Разбиваем множество решений и производим оценку:

```
----------------------------
```

Оценка узла   = 22.000000
Оценка {1,4}   = 29.000000
Оценка не {1,4} = 29.000000
Оценка множества ребер, содержащих [1-4]
и ребер, не содержащих [1-4] равны, поэтому выбирать можно
любое.

Выберем путь [1-4] ...
Вычеркиваем строку 1 и столбец 4.
```
  0.0   1.0   2.0   3.0
  2.0   0.0     *   0.0
  3.0   1.0   0.0     *
  4.0     *   1.0   0.0
```
Самый "тяжелый" нуль получен в строке 3, столбце 2.
Разбиваем множество решений и производим оценку:

Оценка узла   = 29.000000
Оценка {3,2}   = 29.000000
Оценка не {3,2} = 31.000000

Оценка множества ребер, содержащих [3-2] меньше,
поэтому выбираем путь [3-2] ...
Вычеркиваем строку 3 и столбец 2.
```
  0.0   1.0   3.0
  2.0   0.0     *
  0.0   1.0   3.0
  2.0   0.0     *
  4.0     *   0.0
```
Самый "тяжелый" нуль получен в строке 2, столбце 1.
Разбиваем множество решений и производим оценку:

Оценка узла   = 29.000000
Оценка {2,1}   = 29.000000
Оценка не {2,1} = 29.000000

Оценка множества ребер, содержащих [2-1]
и ребер, не содержащих [2-1] равны, поэтому выбирать можно любое.
Выберем путь [2-1] ...
Вычеркиваем строку 2 и столбец 1.
Остался единственный путь [4-3]. Выбираем его.
Алгоритм поиска оптимального пути успешно завершен.
Один из возможных поточечных обходов следующий:
```
  1  - 2  - 3  - 4  -   1.
```

Варианты заданий.

№1	x	10	20	25	40	40	60	70	60	90	30	40	80	75	60	110
	y	130	140	125	140	120	140	130	110	120	60	55	90	45	20	45

№2	x	40	70	90	80	10	15	20	10	95	90	70	60	10	30	5	20
	y	140	120	120	110	90	80	70	60	70	60	60	55	45	45	30	10

№3	x	130	120	110	110	30	5	20	45	60	75	80	30	80	70	90	80
	y	25	15	30	45	40	30	15	20	90	95	90	60	140	130	120	110

№4	x	20	10	40	70	70	120	110	120	5	10	35	65	120	110	105	130	40	80	60
	y	70	60	55	95	85	80	70	60	30	20	20	40	40	30	20	25	120	140	110

№5	x	20	35	60	75	95	90	70	40	10	30	40	140	130	110	130	105	120
	y	10	20	45	45	40	110	130	120	45	60	53	120	110	50	45	20	15

№6	x	30	40	60	95	85	75	70	65	110	120	10	20	15	30	40	60	70
	y	60	50	90	60	65	45	30	5	10	15	90	95	80	80	120	140	130

№7	x	80	90	110	110	120	10	25	20	40	15	10	20	75	95	70	60	85	90
	y	110	120	130	140	120	125	130	125	120	90	80	95	45	40	30	30	5	5

№8	x	40	20	25	10	65	80	95	110	130	120	60	55	65	50	70
	y	70	75	70	60	10	15	20	50	45	55	90	85	80	95	85

№9	x	10	30	20	5	20	10	110	125	140	140	140	120	25	40	60	20	10
	y	45	45	40	30	25	15	60	65	80	140	120	120	125	140	140	130	125

№10	x	120	110	115	25	30	40	40	15	30	40	35	140	120	105	110	125
	y	125	130	120	120	140	140	140	90	85	95	90	55	60	60	70	65

№11	x	40	50	60	110	120	130	103	110	10	20	25	40	10	30	25	20	30
	y	35	40	45	10	15	25	20	30	130	140	125	140	60	55	50	60	65

№ 12	x	30	40	25	110	130	120	125	110	130	120	110	135	10	20	25	40	40	20
	y	45	50	40	140	130	135	140	45	45	40	30	30	130	135	125	140	125	145

№13	x	15	30	40	20	40	10	25	30	30	95	80	90	70	5	10	20	20	35
	y	80	80	90	70	75	135	140	125	140	90	90	80	85	30	15	25	10	20

№ 14	x	45	45	40	40	30	25	15	10	20	110	110	130	120	90	110	110	120	130
	y	10	30	20	35	5	130	120	110	125	70	85	95	80	120	140	130	120	120

№15	x	55	60	60	70	70	70	70	90	90	120	130	140	130	5	20	20	10	5
	y	40	30	10	40	20	95	85	80	60	120	130	140	110	30	25	10	15	30

№	x	120	140	110	125	105	60	65	80	90	40	40	60	80	70	60	110	105	110	120	130	130
16	y	80	80	70	70	60	20	5	10	20	140	120	140	140	130	110	30	20	10	5	25	45

№17	x	70	70	60	60	80	70	80	90	90	85	5	15	10	20	110	120	115	120
	y	125	110	105	120	140	130	90	95	80	70	20	20	150	25	50	60	60	65

№18	x	130	140	125	140	120	15	20	30	40	20	40	60	70	70	80	85	10	30	20	35	30
	y		20	25	40	40	80	80	80	90	70	70	90	95	100	90	95	45	45	50	50	55

№19	x	60	55	60	70	70	70	60	55	70	80	15	20	10	30	120	130	110	120
	y	120	140	110	125	130	80	70	60	60	80	10	35	20	5	40	45	30	40

№20	x	15	30	40	20	40	10	30	20	35	40	110	110	120	130	120	130	125	130
	y	70	60	80	75	75	40	45	40	45	50	140	130	120	110	10	25	15	20

№21	x	45	40	45	40	70	60	65	80	90	20	25	40	40	60	20	140	140	130	130
	y	60	65	70	80	30	20	5	10	20	140	125	140	120	140	140	120	140	130	120

№22	x	5	5	10	5	15	70	80	75	60	100	105	110	120	125	125	120	130
	y	5	10	15	20	20	10	15	15	20	90	100	85	90	105	105	60	65

№23	x	120	140	105	125	30	40	60	40	10	20	110	115	120	130	65	60	70	65	70
	y	60	55	60	70	60	55	90	70	60	70	105	110	100	115	5	20	15	25	30

№24	x	30	20	35	5	20	10	20	35	110	120	115	125	140	95	90	80	70	60
	y	45	40	40	3	25	15	10	20	60	50	65	60	50	90	80	90	95	55

№25	x	55	50	70	65	70	10	20	15	30	40	20	40	100	130	120	140	80	90	60
	y	140	130	120	135	130	90	90	80	80	80	70	70	80	90	90	95	10	20	20

6.6 Лабораторная работа №6. Решение задачи безусловной оптимизации функций одной переменной

Требуется найти минимум заданной функции одной переменной методами равномерного поиска, золотого сечения и методом Ньютона.

Пример.

Пусть задана функция одной переменной $f(x)=x^4+2x^2+4x+1$. Требуется найти минимум функции на интервале [-10,10] с заданной точностью eps=0.001.

6.6.1 Метод равномерного поиска

Метод реализован в процедуре Search.

Входные параметры:

- минимизируемая функция f типа FType, где FType=Function (x: Real): Extended ;
- интервал неопределенности *[a,b]*, на котором будет осуществляться поиск;
- точность вычислений E;

Выходные параметры:

- интервал, в котором лежит искомое значение *x*;
- f_{min} -минимальное значение функции ;
- к- количество итераций .

Текст программы.

```
Function Search(a,b : Real; N : Integer; f : FType;var Res : Byte; E : Real)
: Extended;
        Var Setka : Array of Real; //Содержит сетку узлов
    h    : Real;        //Шаг
    I    : Integer;
    min  : Extended;        //Значение минимального элемента
    k    : Integer;      //Номер минимального элемента
  Begin
  If N<=2 Then Begin  //Недопустимое число узлов сетки, должно быть
  больше 2-ух
          Res:=1;//Код ошибки
          Result:=0;
          Exit;
```

```
                End
                Else
                Begin
                Res:=0;//Ошибки нет
                End;
        If b<=a Then Begin  //Границы заданы неверно
                Res:=2;//Код ошибки
                Result:=0;
                Exit;
                End;

h:=(b-a)/(N-1); //Вычисления шага

SetLength(Setka,N);

Setka[0]:=a;Setka[N-1]:=b;      //Заполнение узлов сетки
For I:=1 to N-2 Do Setka[I]:=a+h*I;//Заполнение узлов сетки

min:=f(Setka[0]);k:=0;                  //Поиск минимального элемента
for I:=1 to N-1 do if f(Setka[I])<min Then Begin min:=f(Setka[I]);k:=I;End;

If (k=0) OR (k=N-1) Then Begin Result:=Setka[k];Exit;End;
                //Минимум находится на границе
IterCount:=1;
  While (Setka[k+1]-Setka[k-1])>E do   //Пока требуемая точность не дос-
тигнута делать ...
    Begin
    a:=Setka[k-1];  //Переход к новым границам
    b:=Setka[k+1];
    h:=(b-a)/(N-1); //Вычисления шага
    Setka[0]:=a;Setka[N-1]:=b;      //Заполнение узлов сетки
    For I:=1 to N-2 Do Setka[I]:=a+h*I;//Заполнение узлов сетки

    min:=f(Setka[0]);k:=0;                  //Поиск минимального элемента
    for I:=1 to N-1 do if f(Setka[I])<min Then Begin min:=f(Setka[I]);k:=I;End;
  Inc(IterCount);
  End;//Конец цикла
Result:=Setka[k];
End;
```

Результат работы программы.

Полученный интервал равен [-0.688;-0.680].

f_{min}=-0.581

к=2

6.6.2 Метод золотого сечения

Метод реализован в процедуре Gold_SechEx.

Входные параметры:

- минимизируемая функция f типа FType, где FType=Function(x : Real) : Extended ;
- интервал неопределенности *[a,b]*, на котором будет осуществляться поиск;
- точность вычислений E;

Выходные параметры:

- RES - код ошибки;
- интервал, в котором лежит искомое значение *x*;
- f_{min} -минимальное значение функции;
- к- количество итераций.

Текст программы.

```
Function GoldSechEx(a,b : Real;f : FType;var Res : Byte; E : Real): Real;
var c,d : Real;       //c,d - Точки деления

Begin
If b<=a Then Begin  //Границы заданы неверно
          Res:=1;//Код ошибки
          Result:=0;
          Exit;
          End
          Else
          Res:=0;
c:=a+0.382*(b-a);//Задание точек деления
d:=a+0.618*(b-a);//Задание точек деления
If f(c)>f(d) Then  //Проверка условий теоремы
          Begin
             a:=c;     //Вычисление новых границ и точек деления
             c:=d;
             d:=c+0.382*(b-c);
          End
          Else
          Begin
             b:=d;     //Вычисление новых границ и точек деления
             d:=c;
             c:=a+0.318*(b-a)
          End;
```

```
  IterCount:=1;
  While (b-a)>E do
          Begin
          If f(c)>f(d) Then  //Проверка условий теоремы
                  Begin
                    a:=c;     //Вычисление новых границ и точек деления
                    c:=d;
                    d:=c+0.382*(b-c);
                  End
                  Else
                  Begin
                    b:=d;     //Вычисление новых границ и точек деления
                    d:=c;
                    c:=a+0.318*(b-a);
                  End;
          Inc(IterCount);
          End;//Конец цикла
  Result:=a+(b-a)/2;
  End; {Function GoldSechEx;}
```

Результат работы программы.

Полученный интервал равен [-0.683;-0.682]

f_{min}=-0.581

К=22

6.6.3 Метод Ньютона

Метод реализован в процедуре Search.

Входные параметры:

- a,b - границы поиска минимума;
- f – функция, которую нужно минимизировать;
- df и ddf – первая и вторая производные соответственно;
- E – требуемая точность;
- X – на входе хранит начальное приближение, на выходе возвращает точку минимума.

Выходные параметры:

- Res – возвращает код ошибки или ноль, если ошибок не было;
- X -точка минимума;
- f_{min} -минимальное значение функции ;
- к- количество итераций .

Текст программы.

```
Function Search(a,b : Real; f,df,ddf : FType;var Res : Byte; E : Real;Var
X : Real) : Extended;
Var I,J : Integer;
   X1 : Real;
Begin
IterCount:=0;
X1:=X;
Repeat
X:=X1;
X1:=X-df(X)/ddf(X);
Inc(IterCount);
Until (Abs(X1-X)<E) Or (X<a) Or (X>b);
Result:=f(X);
End;
```

Результат работы программы.

Полученная точка минимума x=-0,682
f_{min}=-0.581
K=5

Варианты заданий

1 $f(x)=x^2+e^{-0.35x}\to\min$
2 $f(x)=x^2+3e^{-0,45x}\to\min$
3 $f(x)=x^4-1,5\operatorname{arctg}x\to\min$
4 $f(x)=x^2-x+e^{-x}\to\min$
5 $f(x)=-4x+e^{|x-0.2|}\to\min$
6 $f(x)=x-\ln x\to\min$
7 $f(x)=10x\ln x-\frac{x^2}{2}\to\min$
8 $f(x)=e^x+\frac{1}{x}\to\min$
9 $f(x)=x^4-1,1\operatorname{arctg}1,5x\to\min$
10 $f(x)=-2,8x+e^{|x-0,6|}\to\min$
11 $f(x)=\frac{1}{3}x^3-5x+x\ln x\to\min$
12 $f(x)=e^x-\frac{1}{3}x^3+2x\to\min$
15 $f(x)=x\sin x+2\cos x\to\min$
16 $f(x)=-2,2x+e^{|x-0,8|}\to\min$
17 $f(x)=\sqrt{1+x^2}+e^{-2x}\to\min$
18 $f(x)=(x-4)^2+\ln x\to\min$
19 $f(x)=x^2+2e^{-0.65x}\to\min$
20 $f(x)=x^4-0,9\operatorname{arctg}2,5x\to\min$
21 $f(x)=x^4-1,3\operatorname{arctg}\,1,5x\to\min$
22 $f(x)=2x^2+x+\cos^2 x\to\min$
23 $f(x)=x^2+3x(\ln x-1)\to\min$
24 $f(x)=x^4+e^{-x}\to\min$
25 $f(x)=x^2-2x-2\cos x\to\min$
26 $f(x)=x^2+5e^{-0,05x}\to\min$
27 $f(x)=-3,4x+e^{|x-0,4|}\to\min$
28 $f(x)=x^2-2x-e^{-x}\to\min$
29 $f(x)=\cos x/x^2\to\min$
30 $f(x)=e^x+e^{-2x}+2x\to\min$

6.7 Лабораторная работа №7. Решение задачи безусловной оптимизации функций многих переменных

Требуется найти минимум заданной функции многих переменных методами циклического покоординатного спуска и наискорейшего спуска.

Пример.

Пусть задана функция двух переменных $f(\boldsymbol{x}) = 7x_1^2 + 2x_1x_2 + 5x_2^2 + x_1 - 10x_2$, которую необходимо минимизировать с заданной точностью ε=0.001.

6.7.1 Метод циклического покоординатного спуска

Метод реализован в модуле Pokoord_Spusk.pas в функции Function Spusk (E: Real): Real.

Описание типов.

- FType = Function (x: Real): Extended - хранит функцию;
- Point = Array of Real – хранит значения всех переменных;
- MainFunction = Function (P: Point): Extended – целевая функция;
- TGran = Array of Array [1..2] of Real – для хранения границ минимизации каждой переменной.

Используемые глобальные переменные:

- CurrentVar: Integer – содержит номер текущей переменной, по которой производится минимизация;
- P: Point – хранит текущее значение всех переменных;
- Gran: TGran – хранит границы всех переменных;
- FMain: MainFunction – хранит основную функцию;
- IterCount: Integer – хранит количество итераций.

Текст программы.

```
unit Pokoord_Spusk;
{**************************************************************
```

До запуска функции поиска нужно в массив функций f занести все функции по каждой переменной,

в переменную P внести значение начальной точки, а также задать границы для каждой переменной.

Главная функция определяется в вызывающем модуле и записывается в виде "sqr(p[0])+sqr(p[1])", что соответствует функции (x1)^2 + (x2)^2 (предполагается что в массиве P хранятся переменные x1 и x2)

**********}

```
interface
uses Ravnomer_Search;
Type FType = Function(x : Real) : Extended; //Объявляется
                                            //функциональный тип
   Point = Array of Real;  //Тип массив значений всех переменных
   MainFunction = Function(P : Point) : Extended; //Главная функция
   TGran = Array of Array [1..2] of Real;
Var  CurrentVar : Integer;
   P : Point; //Текущая точка
   Gran : TGran;
   FMain : MainFunction;
   IterCount : Integer;
Function Spusk(E : Real) : Real;
        //E - заданная точность
        //FMain - передается главная функция как параметр
implementation
Function NormaV(P1,P2 : Point) : Real; //Нахождение нормы ветора,
                                       //заданного разницей векторов P1 и P2
 Var I : Integer;
 Begin
 Result:=abs(P1[0]-P2[0]);
 For I:=1 to High(P1) do If abs(P1[I]-P2[I])>Result Then Result:=abs(P1[I]-P2[I]);
 End;
 Function Current(x : Real) : Extended;//Функция от конкретной
                                       // переменной
```

```
 Var PTemp : Point;
 Begin
 SetLength(PTemp,High(P)+1);
 PTemp:=Copy(P);
 PTemp[CurrentVar]:=P[CurrentVar]+x;
 Result:=FMain(PTemp);
 End;
Function Spusk;//Поиск минимума главной функции
 var I,J : Integer;
    Neisv : Point; //Массив поправок
    FSub : FType; //Подфункция. Хранит функцию от конкретной
                  // переменной
    Res : Byte;   //Результат поиска минимум функции от конкретной
                  // переменной
 Begin
 SetLength(Neisv,High(P)+1);
 For I:=0 to High(P) do //Перебор всех переменных
  Begin
  CurrentVar:=I;//Установка текущей переменной
  FSub:=Current;//Установка текущей функции
 Neisv[I]:=Search(Gran[i][1],Gran[i][2],100,FSub,Res,E);//Вычисление
                        // минимума функции конкретной переменной
  Neisv[I]:=P[I]+Neisv[I];//Установка новой текущей точки
  End;
 IterCount:=1;
 While NormaV(P,Neisv)>E do//Пока не достигнута нужная точность
  Begin
  For I:=0 to High(P) do//Перебор по всем переменным
    Begin
    For J:=0 to High(P) do P[J]:=Neisv[J];//Переход к новой точке
    CurrentVar:=I;
    FSub:=Current;
    Neisv[I]:=Search(gran[i][1],gran[i][2],100,FSub,Res,E);
```

```
        Neisv[I]:=P[I]+Neisv[I];
        End;
      Inc(IterCount);
      End;
    Result:=FMain(Neisv);//Вычисление значения главной функции
                          // в точке минимума
    End;
   end.
```

Результат работы программы.

Наименьшее значение функция достигает в точке $\boldsymbol{x} = (-0.221; 1.044)$, $f(\boldsymbol{x}) = -5.331$.

6.7.2 Метод наискорейшего спуска

Метод реализован в модуле Naiskor_Spusk.pas в функции Function Spusk (E: Real): Real.

Описание типов.

- FType = Function(x : Real) : Extended - хранит функцию;
- Point = Array of Real – хранит значения всех переменных;
- MainFunction = Function(P : Point) : Extended – целевая функция;
- TGran = Array of Array [1..2] of Real – для хранения границ минимизации каждой переменной;
- TFunction = Array of FType – содержит производные по всем переменным.

Используемые глобальные переменные:

- CurrentVar : Integer – содержит номер текущей переменной, по которой производится минимизация;
- FPr : TFunction – хранит все производные;
- P : Point – хранит текущее значение всех переменных;
- Gran : TGran – хранит границы всех переменных;
- Grad : Point – вектор-градент;
- FMain : MainFunction – хранит основную функцию;
- IterCount : Integer хранит количество итераций.

Текст программы.

```
unit Naiskor_spusk;
{**********************************************************
До запуска функции поиска нужно в массив функций f занести все функ-
ции по каждой переменной,
в переменную FPr внести производные по каждой переменной,
в переменную P внести значение начальной точки,
а также задать границы для каждой переменной.
Главная функция определяется в вызывающем модуле и записывается в
виде
 "sqr(p[0])+sqr(p[1])", что соответствует функции (x1)^2 + (x2)^2
**********************************************************}

interface
uses Ravnomer_Search;
Type
    Point = Array of Extended;  //Тип массив значений всех переменных
    FType = Function(x : Real) : Extended; //Объявляется функциональный
                                           // тип
    TFunction = Array of FType; //Тип массив функций.
                                // Хранит функции от каждой переменной
    MainFunction = Function(P : Point) : Extended; //Главная функция
    TGran = Array of Array [1..2] of Real;
Var  CurrentVar : Integer;
    FPr : TFunction;//Содержит производные по каждой переменной
    P : Point; //Текущая точка
    Gran : TGran; //Границы
    grad : Point;  //Градиентный вектор
    FMain : MainFunction;
    IterCount : Integer;
Function Spusk(E : Real) : Real;
          //E - заданная точность
          //FMain - передается главная функция как параметр

implementation

Function NormaV(P1,P2 : Point) : Real; //Нахождение нормы ветора,
                        //заданного разницей векторов P1 и P2
Var I : Integer;
Begin
Result:=abs(P1[0]-P2[0]);
For I:=1 to High(P1) do If abs(P1[I]-P2[I])>Result Then Result:=abs(P1[I]-
P2[I]);
```

```
End;

Function Current(x : Real) : Extended;//Функция от конкретной переменной
Var PTemp : Point;
Begin
SetLength(PTemp,High(P)+1);
PTemp:=Copy(P);
PTemp[CurrentVar]:=P[CurrentVar]+x*Grad[CurrentVar];
Result:=FMain(PTemp);
End;

Function Spusk;//Поиск минимума главной функции
var I,J : Integer;
   Neisv : Point; //Массив поправок
   FSub : FType; //Подфункция. Хранит функцию от конкретной
                 //переменной
   Res : Byte;   //Результат поиска минимума функции от конкретной
                 // переменной
Begin
For I:=0 to High(P) do
 Begin
 Grad[I]:=-FPr[I](P[I]); //Нахождение вектора градиентов
 End;

SetLength(Neisv,High(P)+1);
For I:=0 to High(P) do //Перебор всех переменных
 Begin
 CurrentVar:=I;//Установка текущей переменной
 FSub:=Current;//Установка текущей функции
 Neisv[I]:=Search(Gran[i][1],Gran[i][2],100,FSub,Res,E);//Вычисление ми-
нимум функции конкретной переменной
 Neisv[I]:=P[I]+Grad[I]*Neisv[I];//Установка новой текущей точки
 End;

IterCount:=1;
While NormaV(P,Neisv)>E do//Пока не достигнута нужная точность де-
лать...
 Begin
 For I:=0 to High(P) do//Перебор по всем переменным
   Begin
   For J:=0 to High(P) do P[J]:=Neisv[J];//Переход к новой точке
   CurrentVar:=I;
   FSub:=Current;
   Neisv[I]:=Search(Gran[i][1],gran[i][2],100,FSub,Res,E);
   Neisv[I]:=P[I]+Grad[I]*Neisv[I];
```

```
  End;
 Inc(IterCount);
 End;

Result:=FMain(Neisv);//Вычисление значения главной функции
                     // в точке минимума
End;
end.
```

Результат работы программы.

Наименьшее значение функция достигает в точке $\boldsymbol{x}=(-0.221;1.045)$, $f(\boldsymbol{x})=-5.331$.

Варианты заданий

1. $f(\boldsymbol{x})=7x_1^2+2x_1x_2+5x_2^2+x_1-10x_2\to\min$
2. $f(\boldsymbol{x})=3x_1^2-3x_1x_2+4x_2^2-2x_1+x_2\to\min$
3. $f(\boldsymbol{x})=x_1^2+2x_2^2+e^{x_1^2+x_2^2}-x_1+x_2\to\min$
4. $f(\boldsymbol{x})=\sqrt{x_1^2+x_2^2+1}+\frac{1}{2}x_1-\frac{1}{2}x_2\to\min$
5. $f(\boldsymbol{x})=x_1^2+4x_1x_2+13x_1^2+5x_2\to\min$
6. $f(\boldsymbol{x})=5x_1^2-4x_1x_2+5x_2^2-x_1-x_3\to\min$
7. $f(\boldsymbol{x})=x_1^4+2x_2^4+x_1^2x_2^2+2x_1+x_2\to\min$
8. $f(\boldsymbol{x})=x_1^2+3x_2^2+\cos(x_1+x_2)\to\min$
9. $f(\boldsymbol{x})=\sqrt{1+2x_1^2+x_2^2}+e^{x_1^2+2x_2^2}-x_1-x_2\to\min$
10. $f(\boldsymbol{x})=x_1+5x_2+e^{x_1^2+x_2^2}\to\min$
11. $f(\boldsymbol{x})=4x_1^2+4x_1x_2+6x_2^2-17x_1\to\min$
12. $f(\boldsymbol{x})=2x_1^2-2x_1x_2+3x_2^2+x_1-3x_2\to\min$
13. $f(\boldsymbol{x})=10x_1^2+3x_1x_2+x_2^2+10x_2\to\min$
14. $f(\boldsymbol{x})=x_1^2-2x_1x_2+6x_1^2+x_1-x_2\to\min$

15. $f(\boldsymbol{x})=x_1^4+x_2^4+\sqrt{2+x_1^2+x_2^2}-2x_1+3x_2\to\min$

16. $f(\boldsymbol{x})=2x_1^2+3x_2^2-2\sin\left(\frac{x_1-x_2}{2}\right)+x_2\to\min$

17. $f(\boldsymbol{x})=\ln(1+3x_1^2+5x_2^2+\cos(x_1-x_2)\to\min$

18. $f(\boldsymbol{x})=x_1^2+e^{x_1^2+x_2^2}+4x_1+3x_2\to\min$

19. $f(\boldsymbol{x})=x_1+2x_2+4\sqrt{1+x_1^2+x_2^2}\to\min$

20. $f(\boldsymbol{x})=2x_1-5x_2+e^{x_1^2+\frac{1}{2}x_2^2}\to\min$

21. $f(\boldsymbol{x})=2\sqrt{3+x_1^2+2x_2^2+x_3^2}-x_1-x_3\to\min$

22. $f(\boldsymbol{x})=4x_1^2+5x_2^2+7x_3^2-2x_1x_2+x_1x_3+x_1-x_2+x_3\to\min$

23. $f(\boldsymbol{x})=3x_1^2+4x_2^3+5x_3^2+2x_1x_2-x_1x_3-2x_2x_3+x_1-3x_3\to\min$

24. $f(\boldsymbol{x})=x_1^2+5x_2^2+8x_3^2-x_1x_2+x_1x_3-x_2x_5+5x_1-3x_2+x_3\to\min$

25. $f(\boldsymbol{x})=2x_1^2+4x_2^2+8x_3^2+2x_1x_2-x_1x_3+2x_2x_3+6x_1+7x_3\to\min$

26. $f(\boldsymbol{x})=7x_1^2+4x_2^2+6x_3^2-3x_1x_2+x_1x_3-x_2x_3+x_1-x_2+x_3\to\min$

27. $f(\boldsymbol{x})=5x_1^2+3x_2^2+2x_3^2+2x_1x_2+x_1x_3+x_2x_3+5x_1+x_3\to\min$

28. $f(\boldsymbol{x})=3x_1^2+5x_2^2+4x_3^2+2x_1x_2-x_1x_3-x_2x_3+7x_1+x_3\to\min$

29. $f(\boldsymbol{x})=4x_1^2+4x_2^2+x_3^2-x_1x_2+2x_1x_3+x_2x_3-x_1+x_2-x_3\to\min$

30. $f(\boldsymbol{x})=x_1^2+2x_2^2+x_1^2+x_2^2+x_3+e^{x_2^2+x_3^2}-x_2+x_3\to\min$

6.8 Лабораторная работа №8. Решение задачи нелинейного программирования методами Лагранжа и приведенного градиента Вулфа

При решении использовать методы неопределенных множителей Лагранжа, приведенного градиента Вулфа.

Пример.

Требуется найти максимальное значение заданной целевой функции $f(\boldsymbol{x}) = 3x_1 - 2x_2 - 0.5x_1^2 - x_2^2 + x_1x_2$ при заданных условиях:

$$\begin{cases} 2x_1 + x_2 \le 2 \\ x_1 + 2x_2 \le 2 \\ x_1, x_2 \ge 0 \end{cases}$$

6.8.1 Метод неопределенных множителей Лагранжа

Метод реализован в Lagrang.pas.

Входные данные.

Описание типов.

TKvadrF = Record c : Array of Array of Real; D : Array of Real; End – запись для хранения целевой функции.

TOgr = Record A : Array of Array of Real; B : Array of Real; End запись для хранения системы ограничений.

TSyst = Array of Array of Real – для хранения системы ограничений при переходе к задаче линейного программирования.

Используемые глобальные переменные:

- KF : TKvadrF – хранит целевую функцию;
- Ogr : TOgr – хранит систему ограничений.
- m,n : Integer – количество ограничений и переменных соответственно;
- Syst : TSyst – хранит новую систему ограничений.

Описаны следующие функции.

Функция Function Lagr(Var X : Array of Real;Var Res : Byte) : Real – реализует метод неопределенных множителей Лагранжа. Возвращает оптимальное значение функции.

Параметры функции:

- X – на выходе получает оптимальную точку;
- Res –признак решения (на выходе возвращает 1, если задача решена и ноль, если задача неразрешима).

Процедура Procedure SM (Syst: TSyst; m, n: Integer; Var X: Array of Real; Var Res: Byte) – решает задачу линейного программирования методом искусственного базиса.

Параметры функции:

- Syst – система ограничений;
- m и n – количество ограничений и переменных соответственно;
- X – на выходе содержит оптимальную точку;
- Res – признак решения (на выходе возвращает 1, если задача решена и ноль, если задача неразрешима).

Текст программы.

```
{Метод неопределенных множителей Лагранжа}
{Расчитан на решение задач выпуклого программирования. В вызывающей программе
должна быть определена квадратичная форма KF, ограничения Ogr, а также пере-
менные n и m, хранящие число переменных и ограничений.}
unit Lagrang;
interface
uses dialogs,SysUtils;
Type
  {Тип квадратичная форма.Задается целевая функция}
  TKvadrF = Record
  c : Array of Array of Real;
  D : Array of Real;
  End;
  {Тип ограничений.Задаются ограничения для данной задачи}
  TOgr = Record
  A : Array of Array of Real;
  B : Array of Real;
```

```
    End;

    {Производные функции Лагранжа по всем переменным.}
    TSyst = Array of Array of Real;

Var
  {KF,Ogr,m и n необходимо определить в вызывающей програмее}
  KF : TKvadrF;
  Ogr : TOgr;
  m,n : Integer;//m - количество ограничений,
                //n - количество переменных
  Syst : TSyst;

{Основная функция}
Function Lagr(Var X : Array of Real;Var Res : Byte) : Real;
 //X - в эту переменную вернется оптимальное значение неизвестных
 //Res - вернет 1, если задача решена, 0 - если неразрешима
Procedure SM(Syst : TSyst;m,n : Integer;Var X : Array of Real;Var Res :
Byte);

implementation

Procedure SM;
Var I,J : Integer;
  SystNew : TSyst;//Новая система с большим количеством
                  // переменных чем в Syst
  mm,nn : Integer;//Новое количество переменных и ограничений
  A : Array of Array of Real;//Для хранения всех базисов
 Basis : Array of Integer;//Хранит номера векторов, вошедших в базис
  MV,Cbas,B,BCopy,Ocenki : Array of Real;
            //MV - Хранит коэффициены при каждой переменной в
            //целевой функции
           //CBas - Содержит коэффициенты при соответствующих
           //неизвестных в целевой функции
```

```
                    //B - хранится опорный план
                    //BCopy - временная переменная для вычисления B
                    //Ocenki - хранит оценки
      Optim,ImeetResh : Boolean;
                  //Optim - хранит информацию, оптимален ли план
                  //ImeetResh - Хранит информацию, имеет ли задача решение
      min : Real;//вспомогательнач переменная
      VedStr,VedCol : Integer;
                  //VedCol - ведущий столбец
                  //VedStr - ведущая строка
    Begin
    SetLength(SystNew,n+m,n+m+n+m+n+1);//Определяется размернсть с учетом
                        //дополнительных неизвестных в задаче
    For I:=0 to n+m-1 do For J:=0 to n+m-1 do SystNew[I,J]:=Syst[I,J];
                        //Заносятся старые значения

    For I:=m+n to m+n+n-1 do SystNew[I-m-n,I]:=-1;//Вводятся переменные Vi, для
                        //перехода к равенству в производных по x
    For I:=m+n+n to m+n+n+m-1 do SystNew[I-n-n,I]:=1;//Вводятся переменные Wi, для
                        //перехода к равенству в производных по Lambda
    For I:=m+n+n+m to m+n+n+m+n-1 do SystNew[I-m-n-n-m,I]:=1;
                        //Вводятся переменные
                        //Zi, для введения искусственного базиса
    For I:=0 to n+m-1 do
SystNew[I,n+m+n+n+m+n]:=Syst[I,n+m];//Копируются правые части

    mm:=n+m;            //Определяется новая размерность
    nn:=n+m+n+n+m+n;

    SetLength(A,nn,mm);
    For I:=0 to nn-1 do    //Заполнение матрицы базисов
```

```
  For J:=0 to mm-1 do A[I,J]:=SystNew[J,I];


SetLength(Basis,mm);
For I:=0 to mm-1 do Basis[I]:=nn+I-mm;//В качестве базиса берутся искуственно
                        //введенные вектора, то есть введенные последними
SetLength(MV,nn);
For I:=0 to nn-n-1 do MV[I]:=0;  //Коэффициенты в новой целевой функции отличны
For I:=nn-n to nn-1 do MV[I]:=-1;//от нуля только при искуственных переменных


SetLength(Cbas,mm);
For I:=0 to mm-1 do Cbas[I]:=MV[Basis[I]];//Вводятся коэффициенты при переменных,
                              //входящих в базисное решение
SetLength(B,mm);
For I:=0 to mm-1 do B[I]:=Syst[I,mm];//Начальное опорное решение
SetLength(BCopy,mm);


SetLength(Ocenki,nn);
For I:=0 to nn-1 do  {Вычисление оценок}
  Begin
  Ocenki[I]:=0;
  For J:=0 to mm-1 do Ocenki[I]:=Ocenki[I]+Cbas[J]*A[I,J];
  Ocenki[I]:=Ocenki[I]-MV[I];
  End;


ImeetResh:=True;
For I:=0 to nn-1 do  //Выясняется, есть ли решение у данной задачи
  If Ocenki[I]<0 Then
    Begin
  Optim:=False;//Optim используется как вспомогательная переменная
    For J:=0 to mm-1 do Optim:=Optim Or (A[I][J]>0);
```

```
        ImeetResh:=ImeetResh and Optim;
        End;


      Optim:=True;
      for I:=0 to nn-1 do If Ocenki[I]<0 Then Begin Optim:=False;Break;End;
                                  //Проверка на оптимальность
      If ImeetResh Then//Если имеет решение...
      Begin
       While (Not Optim) do//Пока не оптимально ...
        Begin
        min:=0;
        For I:=1 to nn-1 do//Вычисление минимальной оценки
         If Ocenki[I]<Ocenki[Round(min)] Then min:=I;
        VedCol:=Round(Min);//Ведущей столбец там, где оценка наимень-
шая


        For I:=0 to mm-1 do// Находится первый положительный элемент в
ведущем столбце
         If A[VedCol][I]>0 Then Begin min:=I;Break;End;


        I:=Round(min+1);
        While I<=mm-1 do//Находится ведущая строка.
                        //Она будет там, где отношение базисного
         Begin          //элемента к положительному элементу ведущего
                        //столбца будет минимальным
         If A[VedCol][I]>0 Then
          If (B[I]/A[VedCol][I])<(B[Round(min)]/A[VedCol][Round(min)])
Then Begin min:=I;End;
         Inc(I);
         End;
        VedStr:=Round(min);


        BCopy:=Copy(B);
        For I:=0 to mm-1 do     //Вычисление нового опроного решения
```

```
If I<>VedStr Then// Если строка не ведущая, то вычисляется по
                  // правилу  прямоугольника
        B[I]:=(BCopy[I]*A[VedCol][VedStr]-
        A[VedCol][I]*BCopy[VedStr])/A[VedCol][VedStr]
        Else//Иначе вычисляется делением на ведущий элемент
        B[I]:=BCopy[I]/A[VedCol][VedStr];

For I:=0 to mm-1 do//Вычисление новых векторов A
 Begin
 If I<>VedStr Then//Если строка не ведущая
  For J:=0 to nn-1 do
  Begin//Правило прямоугольника
  A[J][I]:=(A[J][I]*A[VedCol][VedStr]-
  A[VedCol][I]*A[J][VedStr])/A[VedCol][VedStr];
  End;
 End;

{Вычисление новых значений в ведущей строке}
For I:=0 to nn-1 do If I<>VedCol Then
A[I][VedStr]:=A[I][VedStr]/A[VedCol][VedStr];
A[VedCol][VedStr]:=1;
Basis[VedStr]:=VedCol;//Замена в базисе
Cbas[VedStr]:=MV[VedCol];//Новое значение в векторе Cbas

For I:=0 to nn-1 do//Вычисление новых оценок
 Begin
 Ocenki[I]:=0;
 For J:=0 to mm-1 do Ocenki[I]:=Ocenki[I]+Cbas[J]*A[I,J];
 Ocenki[I]:=Ocenki[I]-MV[I];
 End;

Optim:=True;//Проверка плана на оптимальность
for I:=0 to nn-1 do If Ocenki[I]<0 Then Begin Optim:=False;Break;End;
```

```
  End;{While}

  For I:=0 to High(Basis) do If Basis[I]<=n-1 Then X[Basis[I]]:=B[I];
  //  For I:=0 to n-1 do X[I]:=B[I];//Оптимальная точка
 Res:=1;//Задача решена
 End
 Else
 Res:=0;//Задача неразрешима

End;

Function Lagr;
Var I,J : Integer;
Begin
SetLength(Syst,n+m,n+m+1);{n+m так как производные по всем переменным x дают n,  а по всем переменным Lambda дают m. Так как будут храниться и правые части, то вводится +1}
{Заполняется система производных}
For I:=0 to n-1 do
 Begin
 For J:=0 to n-1 do
  if I<>J Then
  Begin
    Syst[I,J]:=-KF.c[i,j];//Минус берется, т.к. в дальнейшем
                          //придется менять знак
  End
  Else
  Begin
   Syst[I,I]:=-2*KF.c[I,I];//Вычисление производных второго порядка.
  End;
 For J:= n to m+n-1 do  {Вычисляются производные подфункций вида Lambda*f(X1..Xn)}
```

```
      Syst[I,J]:=Ogr.A[J-n,I];
  End;

For I:=n to m+n-1 do
  Begin
  For J:= 0 to n-1 do
      Syst[I,J]:=Ogr.A[I-n,J];//Минус не берется, т.к. в дальнейшем
                                //придется менять знак

  For J:=n to m+n-1 do
      Syst[I,J]:=0;
  End;

For I:=0 to n-1 do Syst[I,m+n]:=KF.D[I];//Формируются правые части
                          // системы в произвоных по x
For I:=n to m+n-1 do Syst[I,m+n]:=Ogr.B[I-n];//Формируются правые
                          // части системы в произвоных по Lambda
SM(Syst,m,n,X,Res);//Вызывается симплекс метод для дальнейшего
                   // решения задачи
Result:=0;
For I:=0 to n-1 do//Вычисление оптимального значения функции
  For J:=I to n-1 do
    Result:=Result+KF.C[I,J]*X[I]*X[J];

For I:=0 to n-1 do Result:=Result+KF.D[I]*X[I];
End;
end.
```

Результат работы программы.

Оптимальное решение: $\boldsymbol{x}$=(0.67,0.67).

Оптимальное значение функции: 0,44.

6.8.2 Метод приведенного градиента Вулфа

Метод реализован в модуле Wolf.pas.

Описание типов.

```
TPoint = Array of Real;
TPurposeFunction = Function (P: TPoint): Real;
TVector = Array of Real;
TMatrix = Array of Array of Real;
TProisvod = Array of Function(P : TPoint) : Real.
```

Используемые глобальные переменные.

P: TPoint – хранит текущую точку.
PF: TPurposeFunction – хранит целевую функцию.
Proisvod: TProisvod – массив производных.
d: TVector – служебная переменная.
cc: TMatrix – первая часть квадратичной формы.
dd: TVector – вторая часть квадратичной формы.

Описание функций.

Function Search (n, m: Integer; A: TMatrix): Real – реализует метод Вулфа.

Параметры функции.

m – количество ограничений.
n – количество переменных.
A –матрица коэффициентов системы ограничений.
Текст программы.

```
unit Wolf;
interface
Type TPoint = Array of Real;
    TPurposeFunction = Function(P : TPoint) : Real;
    TVector = Array of Real;
    TMatrix = Array of Array of Real;
    TProisvod = Array of Function(P : TPoint) : Real;
Var P : TPoint;
   PF : TPurposeFunction;
   Proisvod : TProisvod;
   d : TVector;
   cc : TMatrix;
   dd : TVector;
```

```
Function Search(n,m : Integer;A : TMatrix) : Real;

implementation
Uses Ravnomer_Search;

Function Optim(d : TVector) : Boolean;
Var I : Integer;
Begin
Result:=True;
For I:=0 to High(d) do Result:=Result and (d[I]=0);

End;

Procedure ObrMat (N : Byte; A : TMatrix; Var B : TMatrix );
 Var i,j,k : Byte;
     Base  : real;
Begin
 for i:=0 to N-1 do
  for j:=0 to N-1 do   if i=j then B[i][j]:=1  else  B[i][j]:=0;

 for k:=0 to N-1 do
  begin
   Base := A[k][k];
   for i:=0 to N-1 do
   for j:=0 to N-1 do
    begin
     if (i<>k) and (j>k) then  A[i][j] := A[i][j] - A[k][j]*A[i][k]/Base;
     if i<>k then B[i][j] := B[i][j] - B[k][j]*A[i][k]/Base;
    end;
    if Base<>1 then
     for i:=0 to N-1 do
      begin
       A[k][i] := A[k][i]/Base;
       B[k][i] := B[k][i]/Base;
      end;
     for i:=0 to N-1 do if i<>k then  A[i][k] := 0;
  end;
 End;

Function func(x : Real) : Real;
Var I,J : Integer;
Begin
Result:=0;
For I:=0 to High(P)-1 do
```

```
  For J:=I to High(P)-1 do Result:=Result+cc[I,J]*(P[I]+d[I]*x)*(P[J]+d[J]*x);
For I:=0 to High(P)-1 do Result:=Result+dd[I]*(P[I]+x*d[I]);
End;

Function Search;
Var BBObr : TMatrix;
    BB,NN,MatrixTemp : TMatrix;
    I,J,k,l : Integer;
    I1 : Array of Integer;
    I11 : Set of Byte;
    GradF,GradBF,Multipl,rT,dB,dN : TVector;
    Sum,Lambda : Real;
    f : Ravnomer_Search.FType;
    Res : Byte;
    PTemp : TPoint;

Begin
SetLength(BB,m,m);
SetLength(BBObr,m,m);
SetLength(NN,m,n-m);
SetLength(I1,m);
SetLength(PTemp,n);
SetLength(GradF,n);
SetLength(GradBF,m);
SetLength(Multipl,n);
SetLength(rT,n);
SetLength(dB,m);
SetLength(dN,n-m);
SetLength(d,n);
SetLength(MatrixTemp,m,n-m);


f:=func;
PTemp:=Copy(P);
I11:=[];
For I:=0 to m-1 do
  Begin
  I1[I]:=0;
  For J:=0 to n-1 do If PTemp[J]>PTemp[I1[I]] Then Begin
                                        I1[I]:=J;
                                        End;
  PTemp[I1[I]]:=-3000000;
  I11:=I11+[I1[I]];
  End;
k:=0;l:=0;
```

```
For I:=0 to n-1 do
    If I in I11 Then
      Begin
        For J:=0 to m-1 do
        Begin
        BB[J][k]:=A[J][I];
        End;
      Inc(k);
      End
     Else
      Begin
        For J:=0 to m-1 do
        Begin
        NN[J][l]:=A[J][I];
        End;
      Inc(l);
      End;

ObrMat(m,BB,BBObr);

For I:=0 to n-1 do GradF[I]:=Proisvod[I](P);

k:=0;
For I:=0 to n-1 do
  If I in I11 Then
    Begin
    GradBF[k]:=Proisvod[I](P);
    Inc(k);
    End;

For I:=0 to m-1 do
  Begin
  Sum:=0;
  For J:=0 to m-1 do Sum:=Sum+GradBF[J]*BBObr[J][I];
  Multipl[I]:=Sum;
  End;

For I:=0 to n-1 do
  Begin
  Sum:=0;
  For J:=0 to m-1 do Sum:=Sum+Multipl[J]*A[J][I];
  rT[I]:=Sum;
  End;

For I:=0 to n-1 do rT[I]:=GradF[I]-rT[I];
```

```
k:=0;
For I:=0 to n-1 do
  If Not (I in I11) Then
    Begin
    If rT[I]<=0 Then Begin
                dN[k]:=-rT[I];
                End
                Else
                Begin
                dN[k]:=-P[I]*rT[I];
                End;
    Inc(k);
    End;

For I:=0 to m-1 do
  For J:=0 to n-m-1 do
   Begin
   MatrixTemp[I,J]:=0;
   For k:=0 to m-1 do MatrixTemp[I,J]:=MatrixTemp[I,J]+BBObr[I,k]*NN[k,J];
   End;

For I:=0 to m-1 do
  Begin
  dB[I]:=0;
  For J:=0 to n-m-1 do dB[I]:=dB[I]+MatrixTemp[I,J]*dN[J];
  dB[I]:=-dB[I];
  End;

k:=0;l:=0;
For I:=0 to n-1 do
  If I in I11 Then
    Begin
    d[I]:=dB[k];
    Inc(k);
    End
    Else
    Begin
    d[I]:=dN[l];
    Inc(l);
    End;

For I:=0 to n-1 do
  If d[I]<0 Then Begin Lambda:=-P[I]/d[I];Break; End;
```

```
For I:=0 to n-1 do
  If (d[I]<0) and (-P[I]/d[I]<Lambda) Then Lambda:=-P[I]/d[I];

Lambda:=Ravnomer_Search.Search(0,Lambda,100,f,Res,0.001);

For I:=0 to n-1 do P[I]:=P[I]+Lambda*d[I];
For I:=0 to n-1 do If abs(P[I])<0.0001 Then P[I]:=0;

While not Optim(d) do
  Begin

  PTemp:=Copy(P);
  I11:=[];
  For I:=0 to m-1 do
    Begin
    I1[I]:=0;
    For J:=0 to n-1 do If PTemp[J]>PTemp[I1[I]] Then Begin
                                          I1[I]:=J;
                                          End;
    PTemp[I1[I]]:=-3000000;
    I11:=I11+[I1[I]];
    End;
  k:=0;l:=0;
  For I:=0 to n-1 do
     If I in I11 Then
       Begin
         For J:=0 to m-1 do
         Begin
         BB[J][k]:=A[J][I];
         End;
       Inc(k);
       End
      Else
       Begin
         For J:=0 to m-1 do
         Begin
         NN[J][l]:=A[J][I];
         End;
       Inc(l);
       End;

  ObrMat(m,BB,BBObr);

  For I:=0 to n-1 do GradF[I]:=Proisvod[I](P);
```

```
 k:=0;
 For I:=0 to n-1 do
  If I in I11 Then
    Begin
    GradBF[k]:=Proisvod[I](P);
    Inc(k);
    End;

 For I:=0 to m-1 do
  Begin
  Sum:=0;
  For J:=0 to m-1 do Sum:=Sum+GradBF[J]*BBObr[J][I];
  Multipl[I]:=Sum;
  End;

 For I:=0 to n-1 do
  Begin
  Sum:=0;
  For J:=0 to m-1 do Sum:=Sum+Multipl[J]*A[J][I];
  rT[I]:=Sum;
  End;

 For I:=0 to n-1 do rT[I]:=GradF[I]-rT[I];

 k:=0;
 For I:=0 to n-1 do
  If Not (I in I11) Then
    Begin
    If rT[I]<=0 Then Begin
                dN[k]:=-rT[I];
                End
                Else
                Begin
                dN[k]:=-P[I]*rT[I];
                End;
    Inc(k);
    End;

 For I:=0 to m-1 do
  For J:=0 to n-m-1 do
   Begin
   MatrixTemp[I,J]:=0;
   For k:=0 to m-1 do Matrix-
Temp[I,J]:=MatrixTemp[I,J]+BBObr[I,k]*NN[k,J];
   End;
```

```
  For I:=0 to m-1 do
    Begin
    dB[I]:=0;
    For J:=0 to n-m-1 do dB[I]:=dB[I]+MatrixTemp[I,J]*dN[J];
    dB[I]:=-dB[I];
    End;

  k:=0;l:=0;
  For I:=0 to n-1 do
    If I in I11 Then
      Begin
      d[I]:=dB[k];
      Inc(k);
      End
      Else
      Begin
      d[I]:=dN[l];
      Inc(l);
      End;

  For I:=0 to n-1 do
    If d[I]<0 Then Begin Lambda:=-P[I]/d[I];Break; End;

  For I:=0 to n-1 do
    If (d[I]<0) and (-P[I]/d[I]<Lambda) Then Lambda:=-P[I]/d[I];

  Lambda:=Ravnomer_Search.Search(0,Lambda,100,f,Res,0.001);

  For I:=0 to n-1 do P[I]:=P[I]+Lambda*d[I];

  For I:=0 to n-1 do If abs(P[I])<0.0001 Then P[I]:=0;
  For I:=0 to n-1 do If abs(d[I])<0.001 Then d[I]:=0;
  End;{While}
Result:=1;
End;
end.
```

Результат работы программы.

Оптимальное решение: $\boldsymbol{x}$=(0.669,0.671).

Оптимальное значение функции: 0,443.

Варианты заданий.

1. $f(\boldsymbol{x})=-3x_1^2-x_2^2+4x_1+4x_2-2x_1x_2\to\max$

$$\begin{cases}3x_1+4x_2\le 12\\ x_1-2x_2\le 2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

2. $f(\boldsymbol{x})=-3x_1^2-x_2^2+12x_1+4x_2\to\max$

$$\begin{cases}x_1+x_2\le 0\\ \frac{1}{2}x_1+\frac{1}{2}x_2\le -1\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

3. $f(\boldsymbol{x})=-\frac{1}{2}x_1^2-\frac{1}{2}x_2^2+x_1+2x_2\to\max$

$$\begin{cases}2x_1+3x_2\le 2\\ x_1+4x_2\le 5\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

4. $f(\boldsymbol{x})=3x_1-2x_2-\frac{1}{2}x_1^2-x_2^2+x_1x_2\to\max$

$$\begin{cases}2x_1+x_2\le 2\\ x_1+2x_2\le 2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

5. $f(\boldsymbol{x})=3x_1-2x_2-\frac{1}{2}x_1^2-x_2^2+x_1x_2\to\max$

$$\begin{cases}x_1\le 3,\ x_2\le 6\\ x_1\ge 0,\ x_2\ge 0\end{cases}$$

6. $f(\boldsymbol{x})=-4x_1+8x_2-x_1^2-\frac{3}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases}x_1+x_2\le 3\\ x_1-x_2\le 1\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

7. $f(\boldsymbol{x})=-4x_1+8x_2-x_1^2-\frac{3}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases}x_1+x_2\le 1\\ x_1\le 4\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

8. $f(\boldsymbol{x})=-4x_1+8x_2-x_1^2-\frac{3}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases}3x_1+5x_2\le 15\\ x_1-x_2\le 1\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

9. $f(\boldsymbol{x})=3x_1-2x_2-\frac{1}{2}x_1^2-x_2^2+x_1x_2\to\max$

$$\begin{cases}-x_1+2x_2\le 2\\ 2x_1-x_2\le 2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

10. $f(\boldsymbol{x})=-x_1+6x_2-x_1^2-3x_2^2+3x_1 3x_2\to\max$

$$\begin{cases}4x_1+3x_2\le 12\\ -x_1+x_2\le 1\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

11. $f(\boldsymbol{x})=-x_1+6x_2-x_1^2-3x_2+3x_1x_2\to\max$

$$\begin{cases}x_1+x_2\le 3\\ -2x_1+x_2\le 2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

12. $f(\boldsymbol{x})=-x_1+6x_2-x_1^2-3x_2^2+3x_1x_2\to\max$

$$\begin{cases}x_1-x_2\le 0\\ x_2\le 5\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

13. $f(\boldsymbol{x})=6x_2-x_1^2-\frac{3}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases}3x_1+4x_2\le 12\\ -x_1+x_2\le 12\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

14. $f(\boldsymbol{x})=6x_2-x_1^2-\frac{3}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases}-x_1+2x_2\le 2\\ x_1\le 4\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

15. $f(\boldsymbol{x})=6x_2-x_1^2-\frac{3}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases}3x_1+4x_2\le 12\\ -x_1-2x_2\le -2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

16. $f(\boldsymbol{x})=4x_1+4x_2-3x_1^2-x_2^2+2x_1x_2\to\max$

$$\begin{cases}3\mathrm{x}_1+6\mathrm{x}_2\le 18\\ x_1\ge 0,\ x_2\ge 0\end{cases}$$

17. $f(\boldsymbol{x})=8x_1+12x_2-x_1^2-\frac{3}{2}x_2^2\to\max$

$$\begin{cases}-x_1+2x_2\le 2\\ x_1\le 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

18. $f(\boldsymbol{x})=8x_1+12x_2-x_1^2-\frac{3}{2}x_2^2\to\max$

$$\begin{cases}-3x_1+2x_2\le 0\\ 4x_1+3x_2\le 12\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

19. $f(\boldsymbol{x})=3x_1-2x_2-\frac{1}{2}x_1^2-x_2^2+x_1x_2\to\max$

$$\begin{cases}-2x_1-x_2\le -2\\ 2x_1+3x_2\le 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

20. $f(\boldsymbol{x})=6x_1+4x_2-x_1^2-\frac{1}{2}x_2^2-x_1x_2\to\max$

$$\begin{cases}x_1+2x_2\le 2\\ -2x_1+x_2\le 0\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

21. $f(\boldsymbol{x})=6x_1+4x_2-x_1^2-\frac{1}{2}x_2^2-x_1x_2\to\max$

$$\begin{cases}2x_1+x_2\le 2\\ x_2\le 1\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

22. $f(\boldsymbol{x})=6x_1+4x_2-x_1^2-\frac{1}{2}x_2^2-x_1x_2\to\max$

$$\begin{cases}3x_1+2x_2\le 6\\ -3x_1-x_2\le -3\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

23. $f(\boldsymbol{x})=8x_1+6x_2-2x_1^2-2x_2^2\to\max$

$$\begin{cases}-x_1+x_2\le 1\\ 3x_1+2x_2\le 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

24. $f(\boldsymbol{x})=8x_1+6x_2-2x_1^2-x_2^2\to\max$

$$\begin{cases}-x_1+x_2\le 1\\ x_1\le 3\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

25. $f(\boldsymbol{x})=8x_1+6x_2-2x_1^2-x_2^2\to\max$

$$\begin{cases}-x_1+x_2\le 2\\ 3x_1+4x_2\le 12\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

26. $f(\boldsymbol{x})=2x_1+2x_2-x_1^2-2x_2^2+2x_1x_2\to\max$

$$\begin{cases}4x_1+3x_2\le 12\\ x_2\le 3\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

27. $f(\boldsymbol{x})=2x_1+2x_2-x_1^2-2x_2^2+2x_1x_2\to\max$

$$\begin{cases}2x_1+x_2\le 4\\ -x_1+x_2\le 2\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

28. $f(\boldsymbol{x})=2x_1+2x_2-x_1^2-2x_2^2+2x_1x_2\to\max$

$$\begin{cases}2x_1-x_2\le 2\\ x_2\le 4\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

29. $f(\boldsymbol{x})=4x_1+4x_2-3x_1^2-x_2^2+2x_1x_2\to\max$

$$\begin{cases}4x_1+5x_2\le 0\\ 3x_1+2x_2\le 6\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

30. $f(\boldsymbol{x})=8x_1+12x_2-x_1^2-\frac{3}{2}x_2^2\to\max$

$$\begin{cases}-2x_1-x_2\le -4\\ 2x_1+5x_2\le 10\end{cases}$$

$$x_1\ge 0,\ x_2\ge 0$$

6.9 Лабораторная работа №9. Решение задачи нелинейного программирования методом штрафных функций

Рассмотрим пример из п.6.8.

Метод реализован в Straf.pas.

Описание типов.

TPoint = Array of Extended – для хранения вектора значений.

TPurposeFunction = Function (P: TPoint): Real – целевая функция.

TQ = Array of Function (P: TPoint): Real – массив ограничений.

TB = Array of Real – правые части ограничений.

TdF = Array of Function (P: TPoint): Real – производные целевой функции.

TdQ = Array of Array of Function (P: TPoint): Real – производные ограничений.

Используемые глобальные переменные:

- PurposeFunction: TPurposeFunction – хранит целевую функцию;
- IterCount: Integer – хранит количество итераций.

Описаны следующие функции.

Функция Function CurrentStraf (var P: TPoint; L: TPoint; Q: TQ; B: TB; dF: TdF; dQ: TdQ; Lambda: Real; E: Real): Real – реализует метод Штрафных функций. Возвращает оптимальное значение функции.

Параметры функции:

- P – на входе содержит начальную точку, на выходе получает оптимальную точку;
- L – содержит весовые коэффициенты;
- Q – содержит матрицу коэффициентов системы ограничений;
- B – содержит правые части ограничений;
- dF – содержит производные целевой функции;
- dQ – содержит производные ограничений;
- Lambda – содержит штрафной параметр;
- E – задает точность вычислений.

Текст программы.

```
unit Straf;//Метод штрафных функций
{**********************************************************************
************
Целевая функция описывается в вызывающем модуле , переменной
PurposeFunction
присваивается имя функции в вызывающем модуле. Для запуска метода
должны быть
описаны : начальная точка, функции ограничения, правые части ограниче-
ний,
производные целевой функции по каждой переменной, производные каж-
дого ограничения
по каждой переменной, заданы весовые коэф-ты, штрафной параметр
Lambda.
**********************************************************************
************}
interface
Type
    TPoint = Array of Real;  //Тип массив значений всех переменных
    TPurposeFunction = Function(P : TPoint) : Real; //Тип целевая функция
    TQ = Array of Function(P : TPoint) : Real;// Массив функций - ограниче-
ний
    TB = Array of Real;//Массив Bi - правая часть в ограничениях
    TdF = Array of Function(P : TPoint) : Real;//Массив производных целевой
                                              //функции по всем переменным
    TdQ = Array of Array of Function(P : TPoint) : Real;//Производные
                 // каждого ограничения по каждой переменной
Var  PurposeFunction : TPurposeFunction;//Целевая функция
    IterCount : Integer;
{****************************Процедура
поиска*************************************************}
Function CurrentStraf(var P : TPoint;L : TPoint;Q : TQ; B : TB; dF : TdF; dQ :
TdQ; Lambda : Real;E : Real ) : Real;
                //P - Задается начальная точка, на выходе содержит значение
конечной точки
                //L - Весовые коэффициенты
                //Q - Ограничения
                //B - Правые части ограничений
                //dF - Производные целевой функции
                //dQ - Производные ограничений
                //Lambda - штрафной параметр
                //E - заданная точность
implementation
```

```
Function NormaV(P1,P2 : TPoint) : Real; //Нахождение нормы ветора,
                                        //заданного разницей векторов P1 и P2
Var I : Integer;
Begin
Result:=abs(P1[0]-P2[0]);
For I:=1 to High(P1) do If abs(P1[I]-P2[I])>Result Then Result:=abs(P1[I]-
P2[I]);
End;

{***Проверка векторов-градиентов целевой функции и функций-
ограничений на коллинеарность*******}
Function Collin(P : TPoint;dF : TdF; dQ : TdQ; E : Real) : Boolean;
            //P - Текущая точка
            //dF - Производные целевой функции
            //dQ - Производные ограничений
            //E - Степень точности
Var I,J : Integer;
   gradF : Array of Real;//Градиент целевой функции
   gradQ : Array of Array of Real;//Градиенты ограничений
   TT : Array Of Real;
Begin
SetLength(gradF,High(P)+1);
SetLength(TT,High(P)+1);
SetLength(gradQ,High(dQ)+1,High(P)+1);
For I:=0 to High(P) do
  Begin
  gradF[I]:=dF[I](P);//Вычисление градиента целевой функции в текущей
точке
  End;

For I:=0 to High(dQ) do //Цикл по функциям - ограничениям
  Begin
  For J:=0 to High(P) do //Цикл по каждой переменной I-ой функции
    Begin
    GradQ[I][J]:=dQ[I][J](P);//Вычисление градиентов функций-ограничений
в текущей точке
    End;
  End;

{Result:=True;
For I:=0 to High(P) do   //Номер переменной целевой функции
  For J:=0 to High(dQ) do  //Номер ограничения
   For K:=1 to High(P) do    //Номер переменной в J-ом ограничении
   If (GradQ[J][0]=0) OR (GradQ[J][K]=0) Then Begin Result:=False;Exit; End
Else
```

```
    if (GradF[I]/GradQ[J][0])-(GradF[I]/GradQ[J][K])>E Then Result:=False;  }
Result:=True;
For I:=0 to High(P) do   //Номер переменной целевой функции
  For J:=0 to High(dQ) do  //Номер ограничения
    TT[I]:=(GradF[I]/GradQ[J][I]);

For I:=1 To High(TT) do
    if (TT[0]-TT[I])>E Then Result:=False;

End;

Function CurrentStraf;
Var I,J : Integer;
    Li : Real;//Текущий весовой коффициент
    P1,P2 : TPoint;//Новая точка
    dQSum : Real;//Сумма производных ограничений по текущей
                      // переменной в текущей точке
    g : Real;
Begin
SetLength(P1,High(P)+1);
SetLength(P2,High(P)+1);
P2:=Copy(P);

For I:=0 to High(P) do
  Begin

  dQSum:=0;
  For J:=0 to High(Q) do
    Begin
    {Если точка находится в области допустимых решений то текущий ве-
совой коэффеициент равен нулю}
    If B[J]-Q[J](P)>=0 Then Li:=0 Else Li:=L[J];
    dQSum:=dQSum+Li*dQ[J][I](P);//Вычисление второго слагаемого
    End;

  P1[I]:=P[I]+Lambda*(dF[I](P)+dQSum);//Переход к новой точке
  If P1[I]<0 Then P1[I]:=0;//Выбирается максимум из 0 и P1[I]
                                //и заносится в P1[I]

  End;

G:=0;
For I:=0 to High(Q) do G:=G+B[I]-Q[I](P1);
IterCount:=1;
While (Abs(G)>E) And (Not Collin(P1,dF,dQ,E)) do //Пока вектора
                                        //градиенты не будут коллинеарны...
  Begin
  P:=Copy(P1); //Переход к новой текущей точке
```

```
  For I:=0 to High(P) do
    Begin

    dQSum:=0;
    For J:=0 to High(Q) do
      Begin
      If B[J]-Q[J](P)>=0 Then Li:=0 Else Li:=L[J];
      dQSum:=dQSum+Li*dQ[J][I](P);//Вычисление второго слогаемого
      End;

    P1[I]:=P[I]+Lambda*(dF[I](P)+dQSum);//Переход к новой точке
    If P1[I]<0 Then P1[I]:=0;//Выбирается максимум из 0 и P1[I] и заносится
в P1[I]
    End;
  Inc(IterCount);
  G:=0;
  For I:=0 to High(Q) do G:=G+B[I]-Q[I](P1);
  End;
P:=Copy(P1);
Result:=PurposeFunction(P);//Вычисление значения целевой функции
                                  // в точке максимума
End;
end.
```

Результат работы программы.
Оптимальное решение: $\boldsymbol{x}$=(0.670,0.669).
Оптимальное значение функции: 0,442.

Варианты заданий.

1 $f(\boldsymbol{x})=-x_1^2-x_2^2\to\max$
$\begin{cases}(x_1-5)^2+(x_2-5)^2\le 8\\ x_1,x_2\ge 0\end{cases}$

2 $f(\boldsymbol{x})=-x_1^2+x_2^2\to\max$
$\begin{cases}(x_1-4)^2-(x_2-4)^2\le 6\\ x_1,x_2\ge 0\end{cases}$

3 $f(\boldsymbol{x})=-x_1^2+x_2^2\to\max$
$\begin{cases}(x_1-8)^2-(x_2-8)^2\le 16\\ x_1,x_2\ge 0\end{cases}$

4 $f(\boldsymbol{x})=-x_1^4-x_2^4\to\max$
$\begin{cases}(x_1-4)^4-(x_2-4)^4\le 6\\ x_1,x_2\ge 0\end{cases}$

5 $f(\boldsymbol{x})=-x_1^2-x_2^2+2x_1x_2\to\max$
$\begin{cases}(x_1-7)^2-(x_2-7)^2\le 8\\ x_1,x_2\ge 0\end{cases}$

6 $f(\boldsymbol{x})=-x_1^2-x_2^2+2x_1x_2-x_2\to\max$
$\begin{cases}x_1^2-3x_2^2+5x_1x_2\le 2\\ x_1,x_2\ge 0\end{cases}$

7 $f(\boldsymbol{x})=7x_1^2+3x_2^2-3x_1-9x_2+6\to\max$

$$\begin{cases} x_1^3+x_2^2\le 11 \\ 4x_1^2+x_2^2-x_2\le 7 \end{cases}$$

$$x_1,x_2\ge 0$$

8 $f(\boldsymbol{x})=3x_1^2-3x_2^2\to\max$

$$\begin{cases} x_1^3+x_2^4-3x_1^2x_2^2\le 14 \\ x_1^2+5x_2^2-4x_2\le 10 \end{cases}$$

$$x_1,x_2\ge 0$$

9 $f(\boldsymbol{x})=6x_1^5-4x_2^4\to\max$

$$\begin{cases} 2x_1^3-3x_2^2-3x_1^2\le 5 \\ -x_1^2-5x_2^2\le 1 \end{cases}$$

$$x_1,x_2\ge 0$$

10 $f(\boldsymbol{x})=x_1^2-6x_2^2+x_2\to\max$

$$\begin{cases} -5x_1^2+2x_2^2-3x_1+x_2\le -1 \\ x_1^2-6x_2\le 4 \end{cases}$$

$$x_1,x_2\ge 0$$

11 $f(\boldsymbol{x})=7x_1^2-7x_2^2\to\max$

$$\begin{cases} (x_1^2+2x_2^2)^2\le 19 \\ 2x_2^2-4x_2\le 10 \end{cases}$$

$$x_1,x_2\ge 0$$

12 $f(\boldsymbol{x})=-2x_1^2-x_2^2\to\max$

$$\begin{cases} (x_1+2x_2)^2+(x^2-3)^2\le 10 \\ -2x_2^2-4x_2\le -1 \end{cases}$$

$$x_1,x_2\ge 0$$

13 $f(\boldsymbol{x})=x_1^2-7x_2^2+x_2\to\max$

$$\begin{cases} (x_1-3)^2+(x_2-3)^2\le 16 \\ 2x_1^2-4x_2^2\le 8 \end{cases}$$

$$x_1,x_2\ge 0$$

14 $f(\boldsymbol{x})=-x_1^2+x_2^2+x_1x_2\to\max$

$$\begin{cases} (x_1-6)^2+(x_2-6)^2\le 9 \\ 12x_1^2+4x_2^2\le 11 \end{cases}$$

$$x_1,x_2\ge 0$$

15 $f(\boldsymbol{x})=\frac{1}{2}x_1^2-\frac{1}{2}x_2^2+2x_1x_2\to\max$

$$\begin{cases} \frac{1}{2}x_1^2-x_2^2\le 3 \\ 12x_1^2-7x_2^2\le 9 \end{cases}$$

$$x_1,x_2\ge 0$$

16 $f(\boldsymbol{x})=-x_1^4-x_2^4+8x_1x_2\to\max$

$$\begin{cases} (x_1^2-3)^2+(x_2^2-3)^2\le 9 \\ x_1,x_2\ge 0 \end{cases}$$

17 $f(\boldsymbol{x})=\frac{1}{4}x_1^4-2x_1^2+x_2^2-3x_2\to\max$

$$\begin{cases} x_1^2-2x_2^2-x_2\le 6 \\ x_1,x_2\ge 0 \end{cases}$$

18 $f(\boldsymbol{x})=5x_1^2-6x_2^2-3x_1x_2\to\max$

$$\begin{cases} -5x_1^2+2x_2^2-3x_1+x_2\le 7 \\ x_1^2-6x_2^2\le 4 \end{cases}$$

$$x_1,x_2\ge 0$$

20 $f(\boldsymbol{x})=-18x_1^2+6x_2^2-x_1+x_2\to\max$

$$\begin{cases}x_1^2+12x_2^2\le 1\\ x_1,x_2\ge 0\end{cases}$$

19 $f(\boldsymbol{x})=-2x_1^2+x_2^2+17\to\max$

$$\begin{cases}-2x_1^2+6x_2^2-3x_1\le 4\\ -x_1^2+4x_2^2\le 2\\ x_1,x_2\ge 0\end{cases}$$

21 $f(\boldsymbol{x})=-8x_1^2-6x_2^2+\frac{1}{3}x_1x_2-12\to\max$

$$\begin{cases}x_1^3+12x_2^4\le 7\\ 12x_1^2+2x_2^2\le 1\\ x_1,x_2\ge 0\end{cases}$$

22 $f(\boldsymbol{x})=-8x_1^2-6x_2^2\to\max$

$$\begin{cases}x_1^2+12x_2^2-13x_1-7x_2\le 1\\ 2x_1^2+3x_2^2\le 4\\ x_1,x_2\ge 0\end{cases}$$

23 $f(\boldsymbol{x})=-x_1^2-x_2^2+2x_1x_2+4x_1+25\to\max$

$$\begin{cases}x_1^2-3x_2^2+5x_1x_2\le 8\\ 2x_1^2-x_2^2\le 4\\ x_1,x_2\ge 0\end{cases}$$

24 $f(\boldsymbol{x})=11x_1^2+13x_2^2+4x_1-2x_2+6x_1x_2\to\max$

$$\begin{cases}9x_1+x_2\le 18\\ 4x_1^2+x_2^2-x_2\le 1\\ x_1,x_2\ge 0\end{cases}$$

25 $f(\boldsymbol{x})=2x_1^2+x_2^2-2x_2+7x_1x_2\to\max$

$$\begin{cases}9x_1-x_2^2\le 4\\ 4x_1^2+x_2^2\le 4\\ x_1,x_2\ge 0\end{cases}$$

26 $f(\boldsymbol{x})=13x_1^2+8x_2^2-16x_1^2x_2^2-3x_1x_2\to\max$

$$\begin{cases}x_1^2+6x_2^2-2x_1x_2\le 17\\ x_1^2+15x_2^2\le 12\\ x_1,x_2\ge 0\end{cases}$$

27 $f(\boldsymbol{x})=\frac{1}{3}x_1^3-\frac{1}{2}x_2^2-6x_1x_2+x_1-2x_2\to\max$

$$\begin{cases}x_1^2-2x_2^2+x_2\le 3\\ 7x_1-5x_2\le 9\end{cases}$$

$$x_1,x_2\ge 0$$

28 $f(\boldsymbol{x})=4x_1^4+\frac{1}{2}x_2^2+8x_1x_2-3x_1-2x_2\to\max$

$$\begin{cases}x_1^2-2x_2^2+x_2\le 3\\ 2x_1\le -2\end{cases}$$

$$x_1,x_2\ge 0$$

29 $f(\boldsymbol{x})=4x_1^4+\frac{1}{2}x_2^2+8x_1x_2-3x_1+2x_2\to\max$

$$\begin{cases}x_1^2-2x_2^2+x_2\le 3\\ 2x_1\le -2\end{cases}$$

$$x_1,x_2\ge 0$$

30 $f(\boldsymbol{x})=-x_1^2-x_2^2+2x_1x_2+4x_1-2x_2\to\max$

$$\begin{cases}x_1^2-3x_2^2+5x_1x_2\le 8\\ 2x_1^2-x_2^2\le 4\end{cases}$$

$$x_1,x_2\ge 0$$

Список использованной литературы

1. Акулич И. Л. . Математическое программирование в примерах и задачах: Учеб. пособие для студентов эконом. спец. вузов. - М.: Высш. шк., 1986. - 319 с.
2. Амосов А.А., Дубинский Ю.А., Копченова Н.В. Вычислительные методы для инженеров: Учебное пособие. - М.: Высш. шк., 1994. - 554 с.
3. Аттетков А.В., Галкин С.В., Зарубин В.С. Методы оптимизации: Учебник для вузов. - М.: Изд-во МГТУ им. Баумана, 2001. - 440 с.
4. Базара М., Шетти К. Нелинейное программирование. Теория и алгоритмы. – М.: Мир, 1982.-583с.
5. Банди Б. Методы оптимизации. Вводный курс: Пер. с англ. - М.: Радио и связь, 1988.- 128с.
6. Васильев Ф.П. Численные методы решения экстремальных задач. М.: Наука, 1988.-552с.
7. Грешилов А.А. Прикладные задачи математического программирования М.: Изд-во МГТУ им. Н.Э. Баумана, 1990. - 189 с.
8. Данциг Дж. Линейное программирование: Пер. с англ. - М.: Прогресс. 1966. - 600 с.
9. Еремин И.И., Астафьев И.Н. Введение в теорию линейного и выпуклого программирования. — М.: Наука, 1976. - 192 с.
10. Карманов В.Г. Математическое программирование. - М.: Наука, 1975. - 272с.
11. Линейное и нелинейное программирование. /Под ред. проф. И.Н. Ляшенко.-Киев.:ВИЩА ШКОЛА,1975.- 371 с.
12. Мину М. Математическое программирование. Теория и алгоритмы: Пер с франц. - М.: Наука, 1990. - 487 с.
13. Сухарев А.Г., Тимохов А.В., Федоров В.В. Курс методов оптимизации. - М.: Наука, 1986.-328с.
14. Хедли Дж. Нелинейное и динамическое программирование: Пер. с англ. - М.: Мир, 1967.-506с.
15. Юдин Д.Б., Гольштейн Е.Г. Линейное программирование. Теория, методы, приложения. - М.: Наука, 1969. - 424 с.