

**МЕТОДИЧКА ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ
РАБОТЫ №1**

по предмету “Исследование операций”

Черепенников Роман, 3 курс 8 группа

ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ

Линейным программированием (ЛП) называют раздел теории экстремальных задач оптимизации (максимизации или минимизации) линейных функций на множествах конечномерного пространства, которое описывается конечной системой линейных уравнений и неравенств. В общем виде задача линейного программирования выглядит следующим образом:

$$\begin{aligned}c_1 x_1 + c_2 x_2 + \dots + c_n x_n &\rightarrow \max(\min) \\a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n &\leq \alpha_1 \\&\dots \\a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n &\leq \alpha_m \\b_{11} x_1 + b_{12} x_2 + \dots + b_{1n} x_n &= \beta_1 \\&\dots \\b_{k1} x_1 + b_{k2} x_2 + \dots + b_{kn} x_n &= \beta_k\end{aligned}$$

Под **канонической формой** задачи ЛП понимают:

$$\begin{aligned}c_1 x_1 + c_2 x_2 + \dots + c_n x_n &\rightarrow \max \\a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n &= b_1 \\&\dots \\a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n &= b_m \\d_{*1} \leq x_1 \leq d_1^*, \dots, d_{*n} \leq x_n \leq d_n^*\end{aligned}$$

Или в матричной форме:

$$c'x \rightarrow \max, Ax = b, d_* \leq x \leq d^*$$

Функция $c'x$ называется **целевой функцией**, уравнения $Ax = b$ – **основными ограничениями**, а неравенства $d_* \leq x \leq d^*$ – **прямыми ограничениями** задачи.

Наряду с канонической формой широко используется **нормальная задача ЛП**.

$$c'x \rightarrow \max, Ax \leq b, d_* \leq x \leq d^*$$

Обе рассмотренные задачи являются общими, то есть к ним сводится любая задача ЛП.

Симплекс-метод

В дальнейшем рассматривается каноническая форма задачи ЛП.

$$c'x \rightarrow \max, Ax = b, d_* \leq x \leq d^*$$

Каждый вектор $x \in R^n$ удовлетворяющий всем ограничениям задачи называется **планом**. Тогда $X = \{x | Ax = b, d_* \leq x \leq d^*\}$ – **множество планов**.

План x^0 , являющийся решением задачи, т.е. $c'x^0 = \max c'x, x \in X$ называют **оптимальным планом**.

Пусть выполняются условия $X \neq \emptyset$ $\text{rank } A = m$ ($m < n$).

План x называется **базисным**, если выполняются следующие условия:

- 1) $n - m$ координат принимают одно из граничных условий

$$x_j = d_j^* \vee d_{*j}$$

- 2) остальным m координатам x_{j_1}, \dots, x_{j_m} соответствуют линейно независимые векторы условий a_{j_1}, \dots, a_{j_m}

Множество $J_B = \{j_1, \dots, j_m\}$ называется **базисным множеством**

индексов, $J_H = \{1, \dots, n\} \setminus J_B$ – **небазисным**, векторы a_{j_1}, \dots, a_{j_m} –

базисом базисного плана, координаты x_{j_1}, \dots, x_{j_m} – **базисными**

координатами плана, остальные – **небазисными**, а матрица

$A_B = (a_j, j \in J_B)$ называется **базисной матрицей**.

Алгоритм симплекс-метода:

Пусть задан начальный базисный план x с базисным множеством индексов J_B (с базисной матрицей A_B)

1. Вычисляется вектор потенциалов $u' = c'_B A_B^{-1}$
2. Вычисляются небазисные оценки $\Delta_j = c_j - a'_j u, j \in J_H$
3. Проверяются условия оптимальности плана

$$\Delta_j \leq 0 \text{ при } x_j = d_{*j}; \Delta_j \geq 0 \text{ при } x_j = d_j^*, j \in J_H$$

Если они выполняются, то план оптимален и алгоритм завершен. В противном случае переходят к пункту 4

4. Определяется индекс j_0 . Этот индекс можно взять любым из подмножества базисных индексов, для которых не выполняются условия оптимальности.

5. Определяется направление l

$$\begin{aligned} l_{j_0} &= \text{sign} \Delta_{j_0} \\ l_j &= 0, j \in J_H \setminus j_0 \\ A_B l_B &= -a_{j_0} \text{sign} \Delta_{j_0} \end{aligned}$$

6. Шаг θ^0 определяется из условия:

$$\theta^0 = \min\{\theta_{j_0}, \theta_{j_*}\}$$

где

$$\theta_{j_0} = d_{j_0}^* - d_{*j_0}$$

$$\theta_{j_*} = \min\{\theta_j\}, j \in J_B$$

$$\theta_j = \frac{d_j^* - x_j}{l_j}, \text{ при } l_j > 0$$

$$\theta_j = \frac{d_{*j} - x_j}{l_j}, \text{ при } l_j < 0$$

$$\theta_j = \infty, \text{ при } l_j = 0$$

7. Новый план вычисляется по формуле:

$$\bar{x} = x + \theta^0 l$$

В случае если $\theta^0 = \theta_{j_*}$, то базисное множество индексов заменяется следующим образом $\bar{J}_B = (J_B \setminus j_*) \cup j_0$, соответствующим образом изменяется и базисная матрица A_B .

Переходим к шагу 1.

ИСПОЛЬЗОВАНИЕ OR-TOOLS ДЛЯ РЕШЕНИЯ ЗАДАЧ ЛП

Установка OR-Tools

Установка API Google OR-Tools для языка Python выполняется с помощью пакетного менеджера `pip`.

Необходимо в терминале выполнить команду:

```
pip install ortools
```

Пример №1

В качестве первого примера рассмотрим задачу линейного программирования:

$$c'x \rightarrow \max$$

$$x \geq 0$$

$$Ax \leq b$$

где

$$A = \begin{bmatrix} 2 & 1 & -1 & -3 & 4 & 7 \\ 0 & 1 & 1 & 1 & 2 & 4 \\ 6 & -3 & -2 & 1 & 1 & 1 \end{bmatrix}$$

$$b = [7 \quad 16 \quad 6]'$$

$$c = [1 \quad 2 \quad 1 \quad -1 \quad 2 \quad 3]'$$

Для начала необходимо импортировать модуль `pywraplp` из библиотеки OR-Tools, это делается с помощью:

```
from ortools.linear_solver import pywraplp
```

Затем необходимо создать объект `solver`, используемый для решения задач ЛП:

```
solver = pywraplp.Solver.CreateSolver('GLOP')
```

Следующим шагом является добавление переменных в задачу:

```
x1 = solver.NumVar(0, solver.infinity(), 'x1')
```

```
x2 = solver.NumVar(0, solver.infinity(), 'x2')
```

```
x3 = solver.NumVar(0, solver.infinity(), 'x3')
```

```
x4 = solver.NumVar(0, solver.infinity(), 'x4')
```

```
x5 = solver.NumVar(0, solver.infinity(), 'x5')
```

```
x6 = solver.NumVar(0, solver.infinity(), 'x6')
```

После этого в задачу добавляются ограничения:

```
solver.Add(2*x1 + 1*x2 - 1*x3 - 3*x4 + 4*x5 + 7*x6 <= 7)
```

```
solver.Add(0*x1 + 1*x2 + 1*x3 + 1*x4 + 2*x5 + 4*x6 <= 16)
```

```
solver.Add(6*x1 - 3*x2 - 3*x3 + 1*x4 + 1*x5 + 1*x6 <= 6)
```

и целевая функция:

```
solver.Maximize(1*x1 + 2*x2 + 1*x3 - 1*x4 + 2*x5 + 3*x6)
```

Решить задачу можно с помощью метода Solve объекта solver

```
status = solver.Solve()
```

Для вывода результатов решения используется следующий код:

```
if status == pywraplp.Solver.OPTIMAL:
    print('Решение:')
    print(f'Значение целевой функции = {solver.Objective().Value()}')
    print(f'x1 = {x1.solution_value()}')
    print(f'x2 = {x2.solution_value()}')
    print(f'x3 = {x3.solution_value()}')
    print(f'x4 = {x4.solution_value()}')
    print(f'x5 = {x5.solution_value()}')
    print(f'x6 = {x6.solution_value()}')
else:
    print('Задача не имеет решений')
```

Для того чтобы решить задачу целочисленного ЛП необходимо внести следующие изменения.

Solver создается с помощью:

```
solver = pywraplp.Solver.CreateSolver('SCIP')
```

А при добавлении переменных в задачу необходимо использовать IntVar вместо NumVar.

Вывод программы:

Задача ЛП:

Количество переменных: 6

Количество ограничений: 3

Решение:

Значение целевой функции = 27.5

x1 = 0.0

x2 = 11.5

x3 = 4.5

x4 = 0.0

$$x_5 = 0.0$$

$$x_6 = 0.0$$

Задача целочисленного ЛП:

Количество переменных: 6

Количество ограничений: 3

Решение:

Значение целевой функции = 27

$$x_1 = 0$$

$$x_2 = 11$$

$$x_3 = 5$$

$$x_4 = 0$$

$$x_5 = -0$$

$$x_6 = 0$$

Пример №2

Задача:

С трех складов необходимо поставить в пять магазинов сахарный песок в соответствии с заявкой каждого магазина. Объемы запасов песка, имеющегося на складах, объемы заявок магазинов и тарифы на поставку одной тонны груза со складов в магазины даны в транспортных таблицах по вариантам. Найдите оптимальный план поставок.

Объемы запасов песка, объемы заявок магазинов и тарифы на поставку:

Склад / Магазин	M1	M2	M3	M4	M5	Объем запасов
S1	7	9	15	4	18	200
S2	13	25	8	15	5	250
S3	5	11	6	20	12	250
Заявки	80	260	100	140	120	

Построение математической модели:

Введем следующие переменные

x_{ij} – количество сахара, поставленное со склада Si в магазин Mj

p_{ij} – тариф на поставку одной тонны сахара со склада Si в магазин Mj

Z_i – объем запасов на складе Si

R_j – заявка магазина Mj

Тогда задача математически может быть сформулирована следующим образом:

$$\sum_{i,j} x_{ij} p_{ij} \rightarrow \min$$

$$x_{ij} \geq 0$$

$$\sum_j x_{ij} \leq Z_i$$

$$\sum_i x_{ij} = R_j$$

Все шаги программной реализации выполняются аналогично пункту 1.

Исходный код:

```
from ortools.linear_solver import pywraplp
```

```
# тарифы на поставку
```

```
p = [  
    [7, 9, 15, 4, 18],  
    [13, 25, 8, 15, 5],  
    [5, 11, 6, 20, 12],  
]
```

```
# объем запасов на складах
```

```
Z = [200, 250, 250]
```

```
# заявки магазинов
```

```
R = [80, 260, 100, 140, 120]
```



```

# количество магазинов
Nm = len(R)
Ns = len(Z)

solver = pywraplp.Solver.CreateSolver('GLOP')

# добавление переменных задачи
x = []
for i in range(Ns):
    x.append([])
    for j in range(Nm):
        x[i].append(solver.NumVar(0, solver.Infinity(), f'x_{i+1}_{j+1}'))
# проверка того, что добавилось необходимое количество переменных
print(f'Количество переменных: {solver.NumVariables()}')

# добавление ограничений по объемам на складах
for i in range(Ns):
    constrain_expr = [x[i][j] for j in range(Nm)]
    solver.Add(sum(constrain_expr) <= Z[i])
# добавление ограничений по заявкам магазинов
for j in range(Nm):
    constrain_expr = [x[i][j] for i in range(Ns)]
    solver.Add(sum(constrain_expr) == R[j])
# проверка того, что добавилось необходимое количество ограничений
print(f'Количество ограничений: {solver.NumConstraints()}')

# добавление целевой функции
objective_expr = []
for i in range(Ns):
    for j in range(Nm):
        objective_expr.append(x[i][j] * p[i][j])
solver.Minimize(sum(objective_expr))

# решение задачи
status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:
    print(f'Значение целевой функции = {solver.Objective().Value()}')
    solution = [[x[i][j].solution_value() for j in range(Nm)] for i in range(Ns)]

```

```
print('План поставок в виде матрицы:')
for i in range(Ns):
    print(solution[i])
print('')
else:
    print('Задача не имеет решения')
```

Вывод программы:

Количество переменных: 15
Количество ограничений: 8
Значение целевой функции = 5340.0
План поставок в виде матрицы:
[0.0, 60.0, 0.0, 140.0, 0.0]
[30.0, 0.0, 100.0, 0.0, 120.0]
[50.0, 200.0, 0.0, 0.0, 0.0]
]

Исходный код примеров доступен по ссылке:

<https://github.com/charapennikaurm/or-tools-linear-programming>

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Альсевич В.В., Крахотко В.В. Методы оптимизации (конспект лекций) [Электронный ресурс]. – Режим доступа:
<https://elib.bsu.by/bitstream/123456789/99983/2/%d0%a2%d0%b5%d0%bc%d0%b01%28%d0%9b%d0%9f%29.pdf>
2. OR-Tools Python Reference [Electronic resource]. – Mode of access:
https://developers.google.com/optimization/reference/python/index_python
3. OR-Tools Python Reference: Linear Solver [Electronic resource]. – Mode of access:
https://developers.google.com/optimization/reference/python/linear_solver/pywraplp