КУРС "ИССЛЕДОВАНИЕ ОПЕРАЦИЙ"

Лабораторная работа No 1

Тема:
Линейное программирование.
Решение задач оптимизации производства

Вариант 33

Сергиенко Лев 12 группа МСС, 4 курс

Формулировка проблемы

В обувном цехе для изготовления трех моделей обуви используются четыре вида комплектующих материалов, запасы которых, расход их на изготовление одной пары обуви и цены, получаемые от реализации пары обуви, приведены в таблице.

Составьте математическую модель задачи и найдите оптимальное решение.

Запасы сырья и нормы их расхода на производство обуви

Комплектующие	Расход комплектующих на изготовление обуви			Запасы
	сапоги	ботинки	ботильоны	
Кожа	0,2	0,3	0.32	2 700
Каблуки	0.5	0,2	0.45	800
Стельки	0.027	0.022	0.21	1 600
Подошвы	0.1	0.25	0.28	100
Цена ед.. руб.	900	500	700	

Это классическая задача линейного программирования - задача планирования производства при ограниченных ресурсах.

Переменные - объёмы выпуска (действительные, непрерывные), ограничения - линейные по ресурсам, цель - линейная (максимизация прибыли/выручки). Поэтому применимы стандартные методы LP: симплекс, внутренние точки, анализ базиса, исследование вершин многогранника.

Математическая модель ЛП

Переменные:

- x_1 - число пар сапог;
- x_2 - число пар ботинок;
- x_3 - число пар ботильонов.

Целевая функция (максимизация выручки):

$Z = 900x_1 + 500x_2 + 700x_3 -> max$

Ограничения (ресурсы):

$$
\begin{aligned}
0.2x_1 + 0.3x_2 + 0.32x_3 &\le 2700 \\
0.5x_1 + 0.2x_2 + 0.45x_3 &\le 800 \\
0.027x_1 + 0.022x_2 + 0.21x_3 &\le 1600 \\
0.1x_1 + 0.25x_2 + 0.28x_3 &\le 100 \\
x_1,\ x_2,\ x_3 &\ge 0
\end{aligned}
$$

Добавляя $s_1..s_4 \ge 0$, получаем систему равенств:

$$
\begin{aligned}
0.2x_1 + 0.3x_2 + 0.32x_3 + s_1 &= 2700 \\
0.5x_1 + 0.2x_2 + 0.45x_3 + s_2 &= 800 \\
0.027x_1 + 0.022x_2 + 0.21x_3 + s_3 &= 1600 \\
0.1x_1 + 0.25x_2 + 0.28x_3 + s_4 &= 100
\end{aligned}
$$

Это стандартная LP-задача (m=4, n=3).

Листинг программы

Я реализовал прямой симплекс в Python. Код выполняет:

- формирование полной матрицы с искусственными переменными,
- стартовое допустимое решение x0: $x_1, x_2, x_3 = 0,\ s = b$,
- на итерациях вычисляет коэффициенты U решая $A_B^T U = c_B$,
- вычисляет редуцированные стоимости $\Delta_j = c_j - U^T a_j$,
- выбирает входящую переменную с учётом ограничений,
- строит направление изменения переменных,
- обновляет базис и решение.

```
import numpy as np


def simplex(x, basis, A, c, x_min, x_max):
    m, n_vars = A.shape

    # 1. Формирование матрицы базиса и вычисление U
    A_basis = A[:, (basis - 1)]  # m x m
```

```
    c_basis = c[(basis - 1)]  # m

    try:
        U = np.linalg.solve(A_basis.T, c_basis)
    except np.linalg.LinAlgError as e:
        raise RuntimeError("Базисная матрица вырождена, нельзя
продолжить (повторная базовая матрица).") from e

    print("Матрица базиса (строками):")
    for i in range(m):
        print(A_basis.T[i], " ", c_basis[i])
    print("\nU (дуальные):", U)

    # 2. Вычисляем дельты для не базисных переменных
    deltas = {}
    is_minus_exist = False
    for j in range(n_vars):
        if j not in (basis - 1):
            delta = c[j] - np.dot(U, A[:, j])
            # Определяем символ +/-, по вашей логике:
            if delta > 1e-12:
                sign = "-" if x[j] < x_max[j] else "+"
            elif delta < -1e-12:
                sign = "-" if x[j] > x_min[j] else "+"
            else:
                # delta == 0
                sign = "-" if (x[j] > x_min[j] and x[j] < x_max[j]) else
"+"
            deltas[j] = (delta, sign)
            if sign == "-":
                is_minus_exist = True

    print("\nВычисленные дельты и знаки (delta, sign):")
    for j, (delta, sign) in deltas.items():
        print(f" col {j + 1}: delta={delta:.6f}, sign={sign}")

    # Если нет отрицательных, метод сошелся
    if not is_minus_exist:
        print("\nМетод сошёлся: нет входящих переменных. Текущее
решение оптимально.")
        print("X =", x)
        print("Базис =", basis)
        print("Z =", float(np.dot(c, x)))
        return x, basis, 0

    # 3. Выбираем j0 -- первый индекс с sign == "-"
```

```
    j0 = None
    j0_delta = None
    for j, (delta, sign) in deltas.items():
        if sign == "-":
            j0 = j
            j0_delta = delta
            break

    print(f"\nВходящая переменная: j0 = {j0 + 1}, delta =
{j0_delta:.6f}")

    # 4. Формирование направления l (изменение переменных при
увеличении x_j0)

    l = np.zeros(n_vars)
    # логика: если delta>0 => l[j0]=1, иначе -1
    l[j0] = 1.0 if j0_delta > 0 else -1.0
    # остальные не базисные (кроме j0) = 0; базисные будем вычислять
далее
    # Формируем уравнение для изменения базисных переменных: A_basis *
delta_x_basis = - A[:,j0] * l[j0]
    rhs = -A[:, j0] * l[j0]
    # Решаем для изменений базисных переменных
    delta_x_basis = np.linalg.solve(A_basis, rhs)  # m
    # Заполняем l для базисных индексов
    for idx_i, col_index in enumerate((basis - 1)):
        l[col_index] = delta_x_basis[idx_i]

    print("\nВектор направления l (изменения x при увеличении
входящей):")
    for i in range(n_vars):
        print(f" l[{i + 1}] = {l[i]:.6f}", end=", ")
    print()

    # 5. Вычисление допустимых шагов Q для каждой переменной
    Q = np.full(n_vars, np.inf)
    # Для j0 -- может быть ограничено его верхней границей (x_max[j0]
может быть inf)
    # но чаще шаг на входящую тяготеет к ограничениям базисных
переменных
    if np.isfinite(x_max[j0]) and np.isfinite(x_min[j0]):
        Q[j0] = x_max[j0] - x[j0]
    else:
        Q[j0] = np.inf

    for i in range(n_vars):
```

```
        if abs(l[i]) < 1e-15:
            Q[i] = np.inf
            continue
        if l[i] > 0:
            # увеличивая x_j0, переменная i растёт, ограничение --
верхняя граница
            if np.isfinite(x_max[i]):
                Q[i] = (x_max[i] - x[i]) / l[i]
            else:
                Q[i] = np.inf
        else:
            # l[i] < 0, переменная i убывает, ограничение -- нижняя
граница
            if np.isfinite(x_min[i]):
                Q[i] = (x_min[i] - x[i]) / l[i]  # l[i] отрицательное =>
деление даст полож. шаг
            else:
                Q[i] = np.inf

    print("\nКандидатные значения Тета (Q) для ограничений:")
    for i in range(n_vars):
        print(f" Q[{i + 1}] = {Q[i]} ", end=", ")
    print()

    # 6. Выбираем минимальное положительное Q0
    # Отсекаем отрицательные и очень малые
    valid_Q = [(i, Q[i]) for i in range(n_vars) if Q[i] > 1e-12 and
np.isfinite(Q[i])]
    if not valid_Q:
        # Если нет ограничений -- неограничено в улучшении
        raise RuntimeError("Задача неограниченна по направлению
улучшения (unbounded).")
    j_star, Q0 = min(valid_Q, key=lambda t: (t[1], t[0]))

    print(f"\nМинимальный шаг Q0 = {Q0:.6f}, это даст уходящую
переменную j* = {j_star + 1}")

    # 7. Обновление базиса: заменить j_star на j0
    basis_new = basis.copy()
    # заменим элемент равный j_star+1 на j0+1
    replaced = False
    for idx in range(len(basis_new)):
        if basis_new[idx] == j_star + 1:
            basis_new[idx] = j0 + 1
            replaced = True
            break
```

```
    if not replaced:
        # Если уходящая переменная не в базисе (вырождение), то просто
добавим и удалим самый правый
        basis_new = np.append(basis_new, j0 + 1)
        basis_new = np.delete(basis_new, 0)
    basis_new = np.sort(basis_new)
    # 8. Обновляем x
    x_new = x + Q0 * l

    print("\nНовый базис:", basis_new)
    print("Новое решение x:")
    for i in range(n_vars):
        print(f" x[{i + 1}] = {x_new[i]:.6f}", end=", ")
    print("\nТекущее значение целевой функции Z =", float(np.dot(c,
x_new)))

    return x_new, basis_new, 1


A_orig = np.array(
    [
        [0.2, 0.3, 0.32],  # кожа
        [0.5, 0.2, 0.45],  # каблуки
        [0.027, 0.022, 0.21],  # стельки
        [0.1, 0.25, 0.28],  # подошвы
    ]
)
b = np.array([2700.0, 800.0, 1600.0, 100.0])

m, n = A_orig.shape

# Добавляем slack-переменные: всего n_vars = n + m
A = np.hstack([A_orig, np.eye(m)])  # размер m x (n+m)
n_vars = n + m

c = np.concatenate([np.array([900.0, 500.0, 700.0]), np.zeros(m)])

x_min = np.zeros(n_vars)
x_max = np.full(n_vars, np.inf)

# Начальное допустимое решение: x1..x3 = 0, slack = b (s_i = b_i)
x0 = np.concatenate([np.zeros(n), b])

basis0 = np.arange(n + 1, n + m + 1)

print("Матрица A (с добавленными slack):\n", A)
```

```
print("Начальное допустимое x0:", x0)
print("Начальный базис:", basis0)

x = x0.copy()
basis = basis0.copy()
cont = 1
iter_count = 0
while cont and iter_count < 20:
    iter_count += 1
    print("\n======================== Итерация", iter_count,
"========================")
    x, basis, cont = simplex(x, basis, A, c, x_min, x_max)

print("\nИТОГ:")
print("Итераций:", iter_count)
print("Решение x:", x)
print("Базис:", basis)
print("Z =", float(np.dot(c, x)))
```

Результат

Запуск алгоритма дал:
- Статус: Оптимально найдено
- Оптимальное решение (вектор всех переменных: $x_1, x_2, x_3, s_1, s_2, s_3, s_4$):
x = [1000.   0.   0. 2500. 300. 1573.   0.]
То есть:

- $x_1 = 1000$ (сапоги)
- $x_2 = 0$ (ботинки)
- $x_3 = 0$ (ботильоны)
- s1..s4 показаны (s4 = 0 - подошвы исчерпаны)

Оптимальная выручка: Z = 900 000 руб.
Проверка ограничений при $x_1 = 1000, x_2 = x_3 = 0$:

- кожа: $0.2 \cdot 1000 = 200 \leq 2700$
- каблуки: $0.5 \cdot 1000 = 500 \leq 800$
- стельки: $0.027 \cdot 1000 = 27 \leq 1600$
- подошвы: $0.1 \cdot 1000 = 100 = 100$ - подошвы полностью использованы.