```
import pandas as pd
from IPython.display import HTML
```

Текст ссылки#Практика N5 "Транспортная задача с ограничениями на пропускную способность". **Тема:** Оптимизация логистических потоков

Цель: Научиться формулировать и решать транспортную задачу с учетом ограничений на пропускную способность транспортных путей, используя Python и SciPy.

1. Введение

Транспортная задача – классическая задача оптимизации, направленная на минимизацию затрат при доставке ресурсов от поставщиков к потребителям. В реальных логистических цепочках часто возникают **ограничения на пропускную способность** отдельных маршрутов. Учет этих ограничений делает модель более реалистичной и помогает выявлять "узкие места".

Применение в инноватике:

- Оптимизация логистики новых продуктов.
- Распределение ресурсов в R&D.
- Создание новых, эффективных схем поставок.

2. Математическая постановка задачи

Задача: Минимизировать общую стоимость перевозки при соблюдении запасов, потребностей и ограничений на пропускную способность.

Обозначения:

- m – количество поставщиков (заводов)
- n – количество потребителей (регионов)
- x_{ij} – объем перевозки от поставщика i к потребителю j
- a_i – запас поставщика i
- b_j – потребность потребителя j
- c_{ij} – стоимость перевозки единицы груза от i к j
- u_{ij} – пропускная способность маршрута от i к j

Математическая модель:

Минимизировать: $$ Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} $$

При ограничениях:

1. **Запасы:** $$ \sum_{j=1}^{n} x_{ij} \le a_i, \quad i = 1, ..., m $$
2. **Потребности:** $$ \sum_{i=1}^{m} x_{ij} \ge b_j, \quad j = 1, ..., n $$
3. **Пропускная способность:** $$ x_{ij} \le u_{ij}, \quad i = 1, ..., m, \ j = 1, ..., n $$

4. **Неотрицательность:** $$ x_{ij} \ge 0 $$

3. Пример задачи

Сценарий: Инновационная компания производит новый вид электронных компонентов на **двух заводах (m=2)** и доставляет их в **три региональных распределительных центра (n=3)**.

Данные:

- **Запасы (a):**
  - Завод 1 (a_1): 1000 единиц
  - Завод 2 (a_2): 1500 единиц
- **Потребности (b):**
  - РЦ1 (b_1): 800 единиц
  - РЦ2 (b_2): 1200 единиц
  - РЦ3 (b_3): 500 единиц
- **Стоимость перевозки (c_{ij}), $/единица:**

От/К	РЦ1	РЦ2	РЦ3
31	5	7	9
32	6	8	10

- **Пропускная способность маршрутов (u_{ij}), единиц/сутки:**

От/К	РЦ1	РЦ2	РЦ3
31	700	500	400
32	600	800	300

Баланс: Общий запас (1000+1500=2500) равен общей потребности (800+1200+500=2500). Задача сбалансирована.

```
# Импортируем необходимые библиотеки
import numpy as np
from scipy.optimize import linprog

# --- Параметры задачи ---
m = 2 # Количество заводов (поставщиков)
n = 3 # Количество регионов (потребителей)

# Стоимость перевозки c_ij (в порядке: x11, x12, x13, x21, x22, x23)
c = np.array([5, 7, 9, 6, 8, 10])

# Запасы поставщиков a_i
a = np.array([1000, 1500])
```

```
# Потребности потребителей b_j
b = np.array([800, 1200, 500])

# Пропускная способность маршрутов u_ij (в том же порядке, что и c)
u = np.array([700, 500, 400, 600, 800, 300])

print("--- Параметры задачи ---")
print(f"Количество заводов (m): {m}")
print(f"Количество регионов (n): {n}")
print(f"Стоимость перевозки (c): {c}")
print(f"Запасы заводов (a): {a}")
print(f"Потребности регионов (b): {b}")
print(f"Пропускная способность (u): {u}")

# Проверка сбалансированности
total_supply = np.sum(a)
total_demand = np.sum(b)
print(f"\nОбщий запас: {total_supply}")
print(f"Общая потребность: {total_demand}")
if total_supply == total_demand:
    print("Задача сбалансирована.")
else:
    print("Внимание: Задача не сбалансирована! Может потребоваться добавление
фиктивного поставщика/потребителя.")
```

```
--- Параметры задачи ---
Количество заводов (m): 2
Количество регионов (n): 3
Стоимость перевозки (c): [ 5  7  9  6  8 10]
Запасы заводов (a): [1000 1500]
Потребности регионов (b): [ 800 1200  500]
Пропускная способность (u): [700 500 400 600 800 300]

Общий запас: 2500
Общая потребность: 2500
Задача сбалансирована.
```

4. Подготовка ограничений для `scipy.optimize.linprog`

Функция `linprog` ожидает ограничения в определенном формате:

- `c`: Вектор коэффициентов целевой функции (у нас есть).
- `A_ub`, `b_ub`: Матрица и вектор для неравенств вида $Ax \le b$.
- `A_eq`, `b_eq`: Матрица и вектор для равенств вида $Ax = b$.
- `bounds`: Границы для каждой переменной (например, $x_{ij} \ge 0$).

В нашей задаче:

- $m \times n = 2 \times 3 = 6$ переменных x_{ij}.

- $m+n = 2+3=5$ ограничений на запасы и потребности (в сбалансированной задаче это равенства).
- $m \times n = 6$ ограничений на пропускную способность (неравенства).

Переменные x_{ij} будем упорядочивать следующим образом: $(x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23})$.

```
# --- Формирование ограничений ---

# 1. Ограничения по запасам (равенства)
# Каждая строка соответствует одному заводу.
# Столбцы - переменные x_ij в нашем порядке.
A_eq_supply = np.zeros((m, m * n))
for i in range(m):
    A_eq_supply[i, i * n : (i + 1) * n] = 1 # Заполняем 1 для переменных,
исходящих от завода i

b_eq_supply = a

print("Матрица A_eq_supply (запасы):\n", A_eq_supply)
print("Вектор b_eq_supply (запасы):\n", b_eq_supply)

# 2. Ограничения по потребностям (равенства)
# Каждая строка соответствует одному региону.
A_eq_demand = np.zeros((n, m * n))
for j in range(n):
    A_eq_demand[j, j::n] = 1 # Заполняем 1 для переменных, входящих в регион j

b_eq_demand = b

print("\nМатрица A_eq_demand (потребности):\n", A_eq_demand)
print("Вектор b_eq_demand (потребности):\n", b_eq_demand)

# Объединяем ограничения по равенству
A_eq = np.vstack((A_eq_supply, A_eq_demand))
b_eq = np.concatenate((b_eq_supply, b_eq_demand))

print("\nМатрица A_eq (ограничения):\n", A_eq)
print("Вектор b_eq (ограничения):\n", b_eq)

# 3. Ограничения на пропускную способность (неравенства <=)
# Каждая строка соответствует одному маршруту x_ij
A_ub = np.eye(m * n) # Единичная матрица, т.к. каждое x_ij <= u_ij
b_ub = u

print("\nМатрица A_ub (пропускная способность):\n", A_ub)
print("Вектор b_ub (пропускная способность):\n", b_ub)

# 4. Границы переменных (x_ij >= 0)
# linprog по умолчанию предполагает x_ij >= 0, но явно задать тоже можно
# bounds = [(0, None)] * (m * n) # Для каждой из 6 переменных нижняя граница 0,
верхняя - без ограничений
```

```
# Дополнительно: если бы были явные верхние границы, отличные от u_ij,
# их можно было бы учесть в A_ub и b_ub, или использовать bounds=[(0, u_ij)]
# Но в данном случае, u_ij уже учтены как отдельное ограничение <=,
# поэтому достаточно bounds=[(0, None)].
```

```
Матрица A_eq_supply (запасы):
 [[1. 1. 1. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1.]]
Вектор b_eq_supply (запасы):
 [1000 1500]

Матрица A_eq_demand (потребности):
 [[1. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 1.]]
Вектор b_eq_demand (потребности):
 [ 800 1200  500]

Матрица A_eq (ограничения):
 [[1. 1. 1. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1.]
 [1. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 1.]]
Вектор b_eq (ограничения):
 [1000 1500  800 1200  500]

Матрица A_ub (пропускная способность):
 [[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
Вектор b_ub (пропускная способность):
 [700 500 400 600 800 300]
```

5. Решение задачи с помощью `scipy.optimize.linprog`

Функция `linprog` решает задачи линейного программирования. Мы передадим ей:

- `c`: Коэффициенты целевой функции.
- `A_ub`, `b_ub`: Матрицу и вектор для ограничений $\le$.
- `A_eq`, `b_eq`: Матрицу и вектор для ограничений $=$.
- `bounds`: Границы для переменных (в нашем случае $x_{ij} \ge 0$).
- `method='highs'`: Рекомендуемый современный метод для LP.

```
# --- Решение задачи ---
# Переменные: x11, x12, x13, x21, x22, x23

# Объединяем все ограничения-равенства (запасы + потребности)
# Общее число ограничений-равенств: m + n
# Общее число переменных: m * n

# Формируем итоговые матрицы для linprog
# В linprog, если есть и A_eq, и A_ub, они объединяются.
# Наша модель:
# минимизировать c*x
# при A_eq*x = b_eq
# и A_ub*x <= b_ub
# и bounds (x >= 0)

# Убедимся, что все данные корректны
num_vars = m * n
assert len(c) == num_vars
assert A_eq.shape == (m + n, num_vars)
assert len(b_eq) == m + n
assert A_ub.shape == (m * n, num_vars)
assert len(b_ub) == m * n

# Задаем границы для всех переменных: x_ij >= 0
# ( Верхняя граница будет задана через A_ub <= b_ub, поэтому здесь None)
bounds = [(0, None)] * num_vars
print("\nМатрица bounds (ограничения):\n", bounds)
# Вызов linprog
result = linprog(c=c,
                 A_ub=A_ub,
                 b_ub=b_ub,
                 A_eq=A_eq,
                 b_eq=b_eq,
                 bounds=bounds,
                 method='highs') # Используем эффективный метод

# --- Анализ результатов ---
print("--- Результаты оптимизации ---")

if result.success:
    print(f"Оптимальное решение найдено успешно.")
    print(f"Минимальная общая стоимость перевозки: {result.fun:.2f}")

    # Преобразуем плоский вектор x в двумерную матрицу x_ij
    x_optimal = result.x.reshape((m, n))

    print("\nОптимальные объемы перевозок (x_ij):")
    print("       РЦ1   |  РЦ2  |  РЦ3 ")
    print("-------------------------------")
    for i in range(m):
        row_str = f"3{i+1}: "
        for val in x_optimal[i]:
            row_str += f"{val:7.2f} | "
```

```
        print(row_str)

    # --- Проверка ограничений ---
    print("\n--- Проверка ограничений ---")

    # Проверка запасов
    print("Проверка запасов (сумма отправленного <= запас):")
    for i in range(m):
        sent_sum = np.sum(x_optimal[i, :])
        print(f"  Завод {i+1}: Отправлено={sent_sum:.2f}, Макс. запас={a[i]} -
{'OK' if sent_sum <= a[i] else '!!! ВНИМАНИЕ !!!'}")

    # Проверка потребностей
    print("\nПроверка потребностей (сумма полученного >= потребность):")
    for j in range(n):
        received_sum = np.sum(x_optimal[:, j])
        print(f"  Регион {j+1}: Получено={received_sum:.2f}, Мин. потребность=
{b[j]} - {'OK' if received_sum >= b[j] else '!!! ВНИМАНИЕ !!!'}")

    # Проверка пропускной способности
    print("\nПроверка пропускной способности (x_ij <= u_ij):")
    for i in range(m):
        for j in range(n):
            route_val = x_optimal[i, j]
            route_cap = u[i * n + j]
            route_name = f"З{i+1}->РЦ{j+1}"
            print(f"  {route_name}: {route_val:.2f} (max {route_cap}) - {'OK' if
route_val <= route_cap else '!!! ПРЕВЫШЕНО !!!'}")

else:
    print("Ошибка: Не удалось найти оптимальное решение.")
    print(f"Сообщение: {result.message}")
```

```
Матрица bounds (ограничения):
 [(0, None), (0, None), (0, None), (0, None), (0, None), (0, None)]
--- Результаты оптимизации ---
Оптимальное решение найдено успешно.
Минимальная общая стоимость перевозки: 18400.00

Оптимальные объемы перевозок (x_ij):
       РЦ1   |  РЦ2  |  РЦ3
-------------------------------
З1:  400.00 |  400.00 |  200.00 |
З2:  400.00 |  800.00 |  300.00 |

--- Проверка ограничений ---
Проверка запасов (сумма отправленного <= запас):
  Завод 1: Отправлено=1000.00, Макс. запас=1000 - OK
  Завод 2: Отправлено=1500.00, Макс. запас=1500 - OK
```

```
Проверка потребностей (сумма полученного >= потребность):
  Регион 1: Получено=800.00, Мин. потребность=800 - OK
  Регион 2: Получено=1200.00, Мин. потребность=1200 - OK
  Регион 3: Получено=500.00, Мин. потребность=500 - OK

Проверка пропускной способности (x_ij <= u_ij):
  З1->РЦ1: 400.00 (max 700) - OK
  З1->РЦ2: 400.00 (max 500) - OK
  З1->РЦ3: 200.00 (max 400) - OK
  З2->РЦ1: 400.00 (max 600) - OK
  З2->РЦ2: 800.00 (max 800) - OK
  З2->РЦ3: 300.00 (max 300) - OK
```

6. Анализ результатов и выводы

Что мы получили?

- **Минимальная стоимость:** `result.fun` показывает наименьшую возможную стоимость перевозки при соблюдении всех условий.
- **Оптимальные объемы перевозок:** Матрица `x_optimal` показывает, сколько единиц груза нужно отправить по каждому маршруту.

Интерпретация:

- **Маршруты, работающие на пределе ($x_{ij} \approx u_{ij}$):** Это "узкие места" нашей логистической системы. Они ограничивают общую пропускную способность.
- **Маршруты с большим запасом ($x_{ij} \ll u_{ij}$):** Есть потенциал для оптимизации или перераспределения потоков.
- **Выводы для инноваций:**
  - Понимание, какие маршруты критичны.
  - Обоснование для инвестиций в расширение пропускной способности.
  - Создание более эффективных и надежных логистических цепочек.

7. Заключение

Транспортная задача с ограничениями на пропускную способность – важный инструмент для:

- **Реалистичного моделирования** логистических систем.
- **Выявления и устранения "узких мест".**
- **Оптимизации затрат** и повышения эффективности.

Python с библиотеками `SciPy` и `NumPy` делает решение таких задач доступным и практичным.

8. Задачи для самостоятельного решения (для практики)

1. **Измените запасы и потребности:** Сделайте задачу несбалансированной (например, запасы > потребностей). Как это повлияет на постановку и решение? (Возможно, потребуется ввести

фиктивного потребителя).

2. **Измените стоимость перевозки:** Как изменение цен повлияет на оптимальные маршруты?
3. **Уменьшите пропускную способность одного из маршрутов:** Что произойдет с общей стоимостью и распределением потоков?
4. **Добавьте новый завод или регион:** Как изменится постановка и размерность задачи?

```
# Задача 1: Несбалансированная транспортная задача (запасы > потребностей)
import numpy as np
from scipy.optimize import linprog

print("=== ЗАДАЧА 1: НЕСБАЛАНСИРОВАННАЯ ТРАНСПОРТНАЯ ЗАДАЧА ===\n")

# Параметры задачи (запасы > потребностей)
m, n = 2, 3
c = np.array([5, 7, 9, 6, 8, 10])  # Стоимости
a = np.array([1200, 1800])          # Увеличенные запасы
b = np.array([800, 1200, 500])      # Потребности прежние
u = np.array([700, 500, 400, 600, 800, 300])  # Пропускная способность

print(f"Запасы: {a}, Потребности: {b}")
print(f"Общий запас: {sum(a)}, Общая потребность: {sum(b)}")

# Вместо фиктивного потребителя используем неравенства для запасов
A_ub_supply = np.zeros((m, m*n))
for i in range(m):
    A_ub_supply[i, i*n:(i+1)*n] = 1

A_ub_demand = np.zeros((n, m*n))
for j in range(n):
    A_ub_demand[j, j::n] = -1  # Для неравенства >=

A_ub = np.vstack((A_ub_supply, A_ub_demand))
b_ub = np.concatenate((a, -b))  # Обратите внимание на знак для потребностей

# Ограничения пропускной способности
A_ub_capacity = np.eye(m*n)
b_ub_capacity = u

# Объединяем все неравенства
A_ub_total = np.vstack((A_ub, A_ub_capacity))
b_ub_total = np.concatenate((b_ub, b_ub_capacity))

# Решение
result = linprog(c=c, A_ub=A_ub_total, b_ub=b_ub_total,
                 bounds=(0, None), method='highs')

if result.success:
    x_opt = result.x.reshape((m, n))
    print(f"\nМинимальная стоимость: {result.fun:.2f}")
    print("Оптимальные перевозки:")
    print("       РЦ1   РЦ2   РЦ3")
    for i in range(m):
```

```
        print(f"3{i+1}: {x_opt[i,0]:6.1f} {x_opt[i,1]:6.1f} {x_opt[i,2]:6.1f}")

    # Проверка использования запасов
    unused = a - np.sum(x_opt, axis=1)
    print(f"\nНеиспользованные запасы: Завод 1: {unused[0]:.1f}, Завод 2:
{unused[1]:.1f}")
    print(f"Общий неиспользованный запас: {sum(unused):.1f} единиц")
```

```
=== ЗАДАЧА 1: НЕСБАЛАНСИРОВАННАЯ ТРАНСПОРТНАЯ ЗАДАЧА ===

Запасы: [1200 1800], Потребности: [ 800 1200  500]
Общий запас: 3000, Общая потребность: 2500

Минимальная стоимость: 18200.00
Оптимальные перевозки:
       РЦ1   РЦ2   РЦ3
З1:  600.0  400.0  200.0
З2:  200.0  800.0  300.0

Неиспользованные запасы: Завод 1: 0.0, Завод 2: 500.0
Общий неиспользованный запас: 500.0 единиц
```

```
# Задача 2: Изменение стоимости перевозки и анализ чувствительности
import numpy as np
from scipy.optimize import linprog

print("\n=== ЗАДАЧА 2: АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ К СТОИМОСТИ ===\n")

# Исходные параметры
m, n = 2, 3
a = np.array([1000, 1500])
b = np.array([800, 1200, 500])
u = np.array([700, 500, 400, 600, 800, 300])

# Сценарии изменения стоимости
scenarios = [
    ("Базовая", [5, 7, 9, 6, 8, 10]),
    ("Удорожание З1->РЦ1", [8, 7, 9, 6, 8, 10]),
    ("Удешевление З2->РЦ3", [5, 7, 9, 6, 8, 7]),
    ("Изменение всех цен", [6, 6, 8, 7, 7, 9])
]

def solve_transport(costs):
    A_eq_supply = np.zeros((m, m*n))
    for i in range(m):
        A_eq_supply[i, i*n:(i+1)*n] = 1

    A_eq_demand = np.zeros((n, m*n))
```

```
    for j in range(n):
        A_eq_demand[j, j::n] = 1

    A_eq = np.vstack((A_eq_supply, A_eq_demand))
    b_eq = np.concatenate((a, b))
    A_ub = np.eye(m*n)
    b_ub = u

    result = linprog(c=costs, A_ub=A_ub, b_ub=b_ub,
                     A_eq=A_eq, b_eq=b_eq, bounds=(0, None), method='highs')
    return result

for scenario_name, costs in scenarios:
    result = solve_transport(costs)
    if result.success:
        x_opt = result.x.reshape((m, n))
        print(f"{scenario_name}:")
        print(f"  Стоимость: {result.fun:.2f}")
        print(f"  Распределение: 31->РЦ1:{x_opt[0,0]:.0f}, 31->РЦ2:
{x_opt[0,1]:.0f}, 31->РЦ3:{x_opt[0,2]:.0f}")
        print(f"                 32->РЦ1:{x_opt[1,0]:.0f}, 32->РЦ2:{x_opt[1,1]:.0f},
32->РЦ3:{x_opt[1,2]:.0f}")
```

```
=== ЗАДАЧА 2: АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ К СТОИМОСТИ ===

Базовая:
  Стоимость: 18400.00
  Распределение: 31->РЦ1:400, 31->РЦ2:400, 31->РЦ3:200
                 32->РЦ1:400, 32->РЦ2:800, 32->РЦ3:300
Удорожание 31->РЦ1:
  Стоимость: 19000.00
  Распределение: 31->РЦ1:200, 31->РЦ2:500, 31->РЦ3:300
                 32->РЦ1:600, 32->РЦ2:700, 32->РЦ3:200
Удешевление 32->РЦ3:
  Стоимость: 17500.00
  Распределение: 31->РЦ1:400, 31->РЦ2:400, 31->РЦ3:200
                 32->РЦ1:400, 32->РЦ2:800, 32->РЦ3:300
Изменение всех цен:
  Стоимость: 17500.00
  Распределение: 31->РЦ1:400, 31->РЦ2:400, 31->РЦ3:200
                 32->РЦ1:400, 32->РЦ2:800, 32->РЦ3:300
```

```
# Задача 3: Уменьшение пропускной способности критического маршрута
import numpy as np
from scipy.optimize import linprog

print("\n=== ЗАДАЧА 3: АНАЛИЗ 'УЗКИХ МЕСТ' ===\n")
```

```
m, n = 2, 3
c = np.array([5, 7, 9, 6, 8, 10])
a = np.array([1000, 1500])
b = np.array([800, 1200, 500])

# Сценарии с разной пропускной способностью
capacity_scenarios = [
    ("Базовая", [700, 500, 400, 600, 800, 300]),
    ("Ограничение З2->РЦ2", [700, 500, 400, 600, 400, 300]),
    ("Сильное ограничение З1->РЦ1", [200, 500, 400, 600, 800, 300]),
    ("Множественные ограничения", [300, 300, 300, 400, 500, 200])
]

def solve_with_capacity(u_values):
    A_eq_supply = np.zeros((m, m*n))
    for i in range(m):
        A_eq_supply[i, i*n:(i+1)*n] = 1

    A_eq_demand = np.zeros((n, m*n))
    for j in range(n):
        A_eq_demand[j, j::n] = 1

    A_eq = np.vstack((A_eq_supply, A_eq_demand))
    b_eq = np.concatenate((a, b))
    A_ub = np.eye(m*n)
    b_ub = u_values

    result = linprog(c=c, A_ub=A_ub, b_ub=b_ub,
                     A_eq=A_eq, b_eq=b_eq, bounds=(0, None), method='highs')
    return result

for scenario_name, u_values in capacity_scenarios:
    result = solve_with_capacity(u_values)
    if result.success:
        print(f"{scenario_name}:")
        print(f"  Минимальная стоимость: {result.fun:.2f}")
        print(f"  Пропускная способность: {u_values}")

        # Анализ использования мощностей
        x_opt = result.x
        utilization = []
        for i in range(len(x_opt)):
            if u_values[i] > 0:
                util = x_opt[i] / u_values[i] * 100
                utilization.append(util)
                if util > 95:
                    print(f"  ВНИМАНИЕ: Маршрут {i+1} используется на
{util:.1f}%!")
```

```
=== ЗАДАЧА 3: АНАЛИЗ 'УЗКИХ МЕСТ' ===
```

```
Базовая:
  Минимальная стоимость: 18400.00
  Пропускная способность: [700, 500, 400, 600, 800, 300]
  ВНИМАНИЕ: Маршрут 5 используется на 100.0%!
  ВНИМАНИЕ: Маршрут 6 используется на 100.0%!
Сильное ограничение З1->РЦ1:
  Минимальная стоимость: 18400.00
  Пропускная способность: [200, 500, 400, 600, 800, 300]
  ВНИМАНИЕ: Маршрут 1 используется на 100.0%!
  ВНИМАНИЕ: Маршрут 2 используется на 100.0%!
  ВНИМАНИЕ: Маршрут 4 используется на 100.0%!
```

```python
# Задача 4: Добавление нового завода
import numpy as np
from scipy.optimize import linprog

print("\n=== ЗАДАЧА 4: РАСШИРЕНИЕ ПРОИЗВОДСТВА ===\n")

# Добавляем третий завод
m, n = 3, 3
c = np.array([5, 7, 9,      # Завод 1
              6, 8, 10,     # Завод 2
              4, 6, 8])     # Новый завод 3 (более низкие стоимости)

a = np.array([1000, 1500, 800])  # Запасы + новый завод
b = np.array([800, 1200, 500])
u = np.array([700, 500, 400,     # Завод 1
              600, 800, 300,     # Завод 2
              500, 600, 400])    # Новый завод 3

print(f"Новые параметры:")
print(f"Заводы: {m}, Регионы: {n}")
print(f"Запасы: {a}")
print(f"Потребности: {b}")
print(f"Стоимости: {c.reshape(3,3)}")

# Формирование ограничений
A_eq_supply = np.zeros((m, m*n))
for i in range(m):
    A_eq_supply[i, i*n:(i+1)*n] = 1

A_eq_demand = np.zeros((n, m*n))
for j in range(n):
    A_eq_demand[j, j::n] = 1

A_eq = np.vstack((A_eq_supply, A_eq_demand))
b_eq = np.concatenate((a, b))
A_ub = np.eye(m*n)
b_ub = u

result = linprog(c=c, A_ub=A_ub, b_ub=b_ub,
```

```
                  A_eq=A_eq, b_eq=b_eq, bounds=(0, None), method='highs')

if result.success:
    x_opt = result.x.reshape((m, n))
    print(f"\nРезультат с новым заводом:")
    print(f"Минимальная стоимость: {result.fun:.2f}")
    print("\nОптимальные перевозки:")
    print("       РЦ1   РЦ2   РЦ3")
    for i in range(m):
        print(f"З{i+1}: {x_opt[i,0]:6.1f} {x_opt[i,1]:6.1f} {x_opt[i,2]:6.1f}")

    # Сравнение с исходной задачей
    original_cost = 18400.00  # Из предыдущего решения
    improvement = original_cost - result.fun
    print(f"\nЭкономия от нового завода: {improvement:.2f}
({improvement/original_cost*100:.1f}%)")
```

```
=== ЗАДАЧА 4: РАСШИРЕНИЕ ПРОИЗВОДСТВА ===

Новые параметры:
Заводы: 3, Регионы: 3
Запасы: [1000 1500  800]
Потребности: [ 800 1200  500]
Стоимости: [[ 5  7  9]
 [ 6  8 10]
 [ 4  6  8]]
```

9. Анализ и выводы по дополнительным задачам

- **Несбалансированные задачи:** Вводятся фиктивные элементы (поставщики/потребители) для выравнивания предложения и спроса, сохраняя сбалансированную структуру LP.
- **Ограничения на уровне потребителей:** Добавляются новые строки в матрицу A_{ub} и b_{ub}, что усложняет задачу, но делает ее более реалистичной.
- **Задачи с фиксированной стоимостью/минимальными партиями:** Требуют перехода к целочисленному или смешанному целочисленному программированию (MILP), что выходит за рамки `linprog`.

Эти примеры демонстрируют, как математическая постановка может быть адаптирована для отражения более сложных реальных сценариев.

10. Задачи для самостоятельного решения

1. **Измените пропускную способность маршрутов:** Попробуйте значительно уменьшить пропускную способность одного из маршрутов (например, до 100 единиц). Как это повлияет на общую стоимость и распределение потоков?
2. **Введите новое ограничение:** Допустим, маршрут З1->РЦ2 теперь имеет ограничение, что по нему можно перевезти **минимум** 300 единиц, если он вообще используется. Как это можно

смоделировать? (Это сложно сделать напрямую в `linprog`, но дает представление о более сложных задачах).

3. **Решите задачу без ограничений на пропускную способность:** Сравните результат с задачей, где эти ограничения учтены. Что это говорит об эффективности маршрутов?

```
# Задача 5: Решение без ограничений пропускной способности
import numpy as np
from scipy.optimize import linprog

print("\n=== ЗАДАЧА 5: СРАВНЕНИЕ С ЗАДАЧЕЙ БЕЗ ОГРАНИЧЕНИЙ ===\n")

m, n = 2, 3
c = np.array([5, 7, 9, 6, 8, 10])
a = np.array([1000, 1500])
b = np.array([800, 1200, 500])
u = np.array([700, 500, 400, 600, 800, 300])

# Решение с ограничениями пропускной способности
A_eq_supply = np.zeros((m, m*n))
for i in range(m):
    A_eq_supply[i, i*n:(i+1)*n] = 1

A_eq_demand = np.zeros((n, m*n))
for j in range(n):
    A_eq_demand[j, j::n] = 1

A_eq = np.vstack((A_eq_supply, A_eq_demand))
b_eq = np.concatenate((a, b))

# С ограничениями
A_ub_with = np.eye(m*n)
b_ub_with = u
result_with = linprog(c=c, A_ub=A_ub_with, b_ub=b_ub_with,
                      A_eq=A_eq, b_eq=b_eq, bounds=(0, None), method='highs')

# Без ограничений пропускной способности
result_without = linprog(c=c, A_eq=A_eq, b_eq=b_eq, bounds=(0, None),
method='highs')

if result_with.success and result_without.success:
    print("Сравнение результатов:")
    print(f"С ограничениями пропускной способности:")
    print(f"  Стоимость: {result_with.fun:.2f}")
    x_with = result_with.x.reshape((m, n))
    print(f"  Распределение: {x_with.flatten()}")

    print(f"\nБез ограничений пропускной способности:")
    print(f"  Стоимость: {result_without.fun:.2f}")
    x_without = result_without.x.reshape((m, n))
    print(f"  Распределение: {x_without.flatten()}")

    cost_difference = result_with.fun - result_without.fun
```

```
    print(f"\nРазница в стоимости: {cost_difference:.2f}")
    print(f"Ограничения увеличили стоимость на
{cost_difference/result_without.fun*100:.1f}%")

    # Анализ "узких мест"
    print(f"\nАнализ ограничивающих факторов:")
    for i in range(m):
        for j in range(n):
            idx = i*n + j
            if x_with[i,j] >= u[idx] * 0.95:  # Используется более 95% capacity
                print(f"  Маршрут З{i+1}->РЦ{j+1}: {x_with[i,j]:.1f}/{u[idx]}
({(x_with[i,j]/u[idx]*100):.1f}%) - ОГРАНИЧИВАЕТ!")
```

```
=== ЗАДАЧА 5: СРАВНЕНИЕ С ЗАДАЧЕЙ БЕЗ ОГРАНИЧЕНИЙ ===

Сравнение результатов:
С ограничениями пропускной способности:
  Стоимость: 18400.00
  Распределение: [400. 400. 200. 400. 800. 300.]

Без ограничений пропускной способности:
  Стоимость: 18400.00
  Распределение: [   0. 1000.    0.  800.  200.  500.]

Разница в стоимости: 0.00
Ограничения увеличили стоимость на 0.0%

Анализ ограничивающих факторов:
  Маршрут З2->РЦ2: 800.0/800 (100.0%) - ОГРАНИЧИВАЕТ!
  Маршрут З2->РЦ3: 300.0/300 (100.0%) - ОГРАНИЧИВАЕТ!
```