Семинар 5

1. Лекция 5: Линейное программирование

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import z3
from matplotlib.patches import Polygon
from ortools.linear_solver import pywraplp
from quantecon.optimize.linprog_simplex import␣
  ↪linprog_simplex
from scipy.optimize import linprog
from sympy.solvers.simplex import lpmax
from sympy import Eq, Symbol
```

1.1. Общая постановка задачи

Задача **линейного программирования** состоит в оптимизации — максимизации или минимизации — некой целевой функции при некоторых ограничениях, выраженных в форме равенств и/или неравенств. Целевая функция должна быть линейной. Также должны быть линейными **все** ограничения.

В задаче линейного программирования оптимизируется (для примера рассмотрим минимизацию) следующая функция

$$c'x = \sum_{i=1}^{n} c_i x_i$$

где $c = (c_1, c_2, \dots, c_n)'$ — единичный вектор цен, $x = (x_1, x_2, \dots, x_n)'$ — вектор целевых переменных.

Целевые переменные должны удовлетворять набору ограничений. Ограничения являются важной частью задачи, так как при их отсутствии (или неверном выборе оных) задача не будет иметь решения.

С учетом ограничений задача будет иметь следующий вид:

$$\begin{aligned}
&\min_x c'x \\
&a_i'x = b_i \quad i \in M_1 \\
&a_i'x \geqslant b_i \quad i \in M_2 \\
&a_i'x \leqslant b_i \quad i \in M_3 \\
&x_j \geqslant 0 \quad j \in N_1 \\
&x_j \leqslant 0 \quad j \in N_2 \\
&x_j \text{ any} \quad j \in N_3
\end{aligned}$$

Вектор x, удовлетворяющий всем ограничениям, называется **допустимым**. Множество допустимых векторов формируют **допустимое множество**.

Соответственно, допустимое решение, минимизирующее целевую функцию, называется **оптимальным**.

Если допустимое множество пусто, то задача **неразрешима**.

Если $\forall K \in \mathbb{R}$ существует допустимое решение x такое, что $c'x < K$, то задача называется **неограниченной** со значением целевой функции $-\infty$.

1.1.1. Пример 1

Рассмотрим следующую задачу: завод выпускает 2 продукта, на производство которых тратится труд и некоторое обобщенное сырье. Продукты продаются по соответствующим ценам.

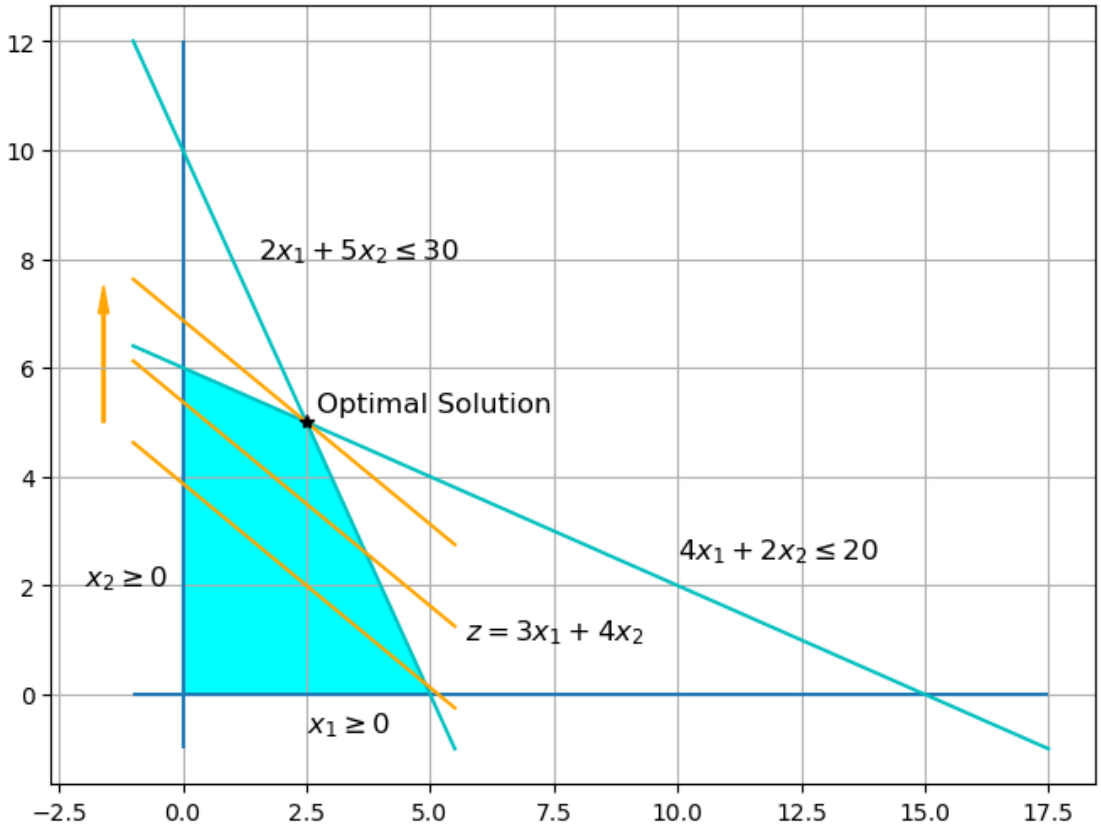
	Продукт 1	Продукт 2
Сырье	2	5
Труд	4	2
Цена	3	4

Предполагается также, что завод располагает 30 единицами сырья и 20 единицами труда.

Итоговая задача, решаемая заводом (предполагая его адекватность и стремление максимизировать прибыль), может быть записана в следующем виде

$$\begin{aligned}
&\max_{x_1, x_2} 3x_1 + 4x_2 \\
&2x_1 + 5x_2 \leqslant 30 \\
&4x_1 + 2x_2 \leqslant 20 \\
&x_1,\ x_2 \geqslant 0
\end{aligned}$$

Допустимое множество и итоговое решение можно показать графически



1.1.2. Пример 2

Рассмотрим инвестиционный фонд, который на протяжении 3 периодов (лет) вкладывает деньги в 3 инструмента:

- Аннуитет. Требует вкладывать одинаковый объем средств в начале каждого периода. В конце 3 периода возвращает 130% от вложенных средств.
- Депозит. В начале каждого периода фонд может положить сумму денег в банк или же занять ее у него. В конце периода фонд получает 106% или выплачивает их. Занять можно не более 20000.
- Облигации. Можно купить только во втором периоде. В конце третьего года фонд получит 130% от вложенных средств. Вложить можно не более 50000.

Начальная сумма — 100000.

Итого имеем следующие переменные:

	Год 1	Год 2	Год 3
Аннуитет	x_1	x_1	x_1
Депозит	x_2	x_3	x_4
Облигации	0	x_5	0

В первый год фонд распределяет начальные 100000

$$x_1 + x_2 = 100000$$

В начале второго года фонд распределяет доходы от депозита между аннуитетом, облигациями и новым депозитом

$$x_1 + x_3 + x_5 = 1.06x_2$$

В начале третьего года фонд распределяет доходы от депозита между аннуитетом и и новым депозитом

$$x_1 + x_4 = 1.06x_3$$

Целевой функцией фонда является

$$Z = 1.30\ 3x_1 + 1.06x_4 + 1.30x_5$$

Итого, задача примет вид

$$\begin{aligned}
\max_x 1.30 \times 3x_1 + 1.06x_4 + 1.30x_5 \\
x_1 + x_2 = 100000 \\
x_1 - 1.06x_2 + x_3 + x_5 = 0 \\
x_1 - 1.06x_3 + x_4 = 0 \\
x_2 \geqslant -20000 \\
x_3 \geqslant -20000 \\
x_4 \geqslant -20000 \\
x_5 \leqslant 50000 \\
x_1,\ x_5 \geqslant 0
\end{aligned}$$

1.2. Стандартная форма задачи

С теоретико-практической точки зрения оптимально перейти от задачи в общем виде, к задаче в **стандартной** форме

$$\begin{aligned}
\min_x c'x \\
a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\
\vdots \\
a_{m1}x_1 + \cdots + a_{mn}x_n = b_m \\
x_j \geqslant 0
\end{aligned}$$

или

$$\begin{aligned}
\min_x c'x \\
Ax = b \\
x \geqslant 0
\end{aligned}$$

Для перевода задачи в стандартную форму можно воспользоваться следующеми правилами:

- Если целевая функция в исходной задаче **максимизируется**, то можно построить к ней аддитивно обратную функцию, минимизация которой будет эквивалентна исходной задаче.
- Если мы имеем ограничение $x_j \leqslant 0$, то можно ввести новую переменную $\tilde{x}_j = -x_j$, для которой српаведливо $\tilde{x}_j \geqslant 0$.
- Если мы имеем неграниченную переменную x_j, то для того, чтобы можно было «наложить» на нее ограничения, введем **две** новые переменные $x_j^+, x_j^- \geqslant 0$ такие, что $x_j = x_j^+ - x_j^-$.
- Если мы имеем ограничение $\sum_{i=1}^n a_{ki}x_i \leqslant b_k$, то его можно привести к равенству при помощи добавления дополнительной переменной $s_k \geqslant 0$ такой, что $\sum_{i=1}^n a_{ki}x_i + s_k = b_k$.

1.2.1. Пример

Вернемся к рассмотренному ранее примеру

$$\begin{aligned} \max_{x_1,x_2} 3x_1 + 4x_2 \\ 2x_1 + 5x_2 \leqslant 30 \\ 4x_1 + 2x_2 \leqslant 20 \\ x_1,\ x_2 \geqslant 0 \end{aligned}$$

Она переписывается в виде

$$\begin{aligned} \max_{x_1,x_2} 3x_1 + 4x_2 \\ 2x_1 + 5x_2 + s_1 = 30 \\ 4x_1 + 2x_2 + s_2 = 20 \\ x_1,\ x_2,\ s_1,\ s_2 \geqslant 0 \end{aligned}$$

1.2.2. Альтернативные стандартные формы

Приведенная выше стандартная форма приводится, например в учебнике Карманова «Математическое программирование» (1986) и книге Sargent и Stachurski. В зарубежных учебниках, например в Sakarovitch «Linear Programming» (1983) и в кладезе мудрости в качестве стандартной формы приводится следующая:

$$\begin{aligned} \max_{x} c'x \\ Ax \leqslant b \\ x \geqslant 0 \end{aligned}$$

1.3. Дуальная проблема

Как гласит литература, каждая задача

$$\begin{aligned} \min_{x} c'x \\ a_i'x = b_i \quad i \in M_1 \\ a_i'x \geqslant b_i \quad i \in M_2 \\ a_i'x \leqslant b_i \quad i \in M_3 \\ x_j \geqslant 0 \quad j \in N_1 \\ x_j \leqslant 0 \quad j \in N_2 \\ x_j \text{ any} \quad j \in N_3 \end{aligned}$$

имеет соответствующую ей **дуальную** задачу

$$\begin{aligned} \max_{x} b'p \\ p_i \text{ any} \quad i \in M_1 \\ p_i \geqslant 0 \quad i \in M_2 \\ p_i \leqslant 0 \quad i \in M_3 \\ A_j'p_j \leqslant c_j \quad j \in N_1 \\ A_j'p_j \geqslant c_j \quad j \in N_2 \\ A_j'p_j = c_j \quad j \in N_3 \end{aligned}$$

Здесь

$$A = \left(\begin{array}{c} a_1' \\ \hline a_2' \\ \hline \vdots \\ \hline a_m' \end{array}\right) = \left(\begin{array}{c|c|c|c} A_1 & A_2 & \cdots & A_n \end{array}\right)$$

Нетрудно заметить, что:

- Каждому ограничению вида $a_i'x = b_i$ ($\leqslant b_i$, $\geqslant b_i$) соответствует переменная p_i.
  - Если $a_i'x \geqslant b_i$, то $p_i \geqslant 0$.
  - Если $a_i'x \leqslant b_i$, то $p_i \leqslant 0$.
  - Если $a_i'x = b_i$, то p_i не ограничивается.
- Каждой переменной $x_1, x_2, \ldots, x_n$ соответствует ограничение.
  - Если $x_j \leqslant 0$, то $A_j'p \geqslant c_j$.
  - Если $x_j \geqslant 0$, то $A_j'p \leqslant c_j$.
  - Если x_j не ограничивается, то $A_j'p = c_j$.

Основная	min	max	Дуальная
Ограничения	$\geqslant b_i$ $\leqslant b_i$ $= b_i$	$\geqslant 0$ $\leqslant 0$ свободно	Знак
Знак	$\geqslant 0$ $\leqslant 0$ свободно	$\leqslant c_j$ $\geqslant c_j$ $= c_j$	Ограничения

The duality theorem has an economic interpretation. If we interpret the primal LP as a classical “resource allocation” problem, its dual LP can be interpreted as a “resource valuation” problem.

Consider a factory that is planning its production of goods. Let x be its production schedule (make x_i amount of good i), let $c \geqslant 0$ be the list of market prices (a unit of good i can sell for c_i). The constraints it has are $x \geqslant 0$ (it cannot produce negative goods) and raw-material constraints. Let b be the raw material it has available, and let $A \geqslant 0$ be the matrix of material costs (producing one unit of good i requires A_{ji} units of raw material j).

Then, the constrained revenue maximization is the primal LP:

> Maximize $c^T x$ subject to $Ax \leqslant b$, $x \geqslant 0$.

Now consider another factory that has no raw material, and wishes to purchase the entire stock of raw material from the previous factory. It offers a price vector of y (a unit of raw material i for y_i). For the offer to be accepted, it should be the case that $A^T y \geqslant c$, since otherwise the factory could earn more cash by producing a certain product than selling off the raw material used to produce the good. It also should be $y \geqslant 0$, since the factory would not sell any raw material with negative price. Then, the second factory’s optimization problem is the dual LP:

> Minimize $b^T y$ subject to $A^T y \geqslant c$, $y \geqslant 0$.

Wikipedia: Dual_linear_program

1.3.1. Задача максимизации: вариант 1

Основной задачей является задача **минимизации**! Поэтому если ваша исходная задача является задачей максимизации, считайте ее **дуальной**. Соответственно, все преобразования будут идти в противоположную сторону!

Вернемся к задаче планирования выпуска

$$\begin{aligned} \max_{x_1,x_2} 3x_1 + 4x_2 \\ 2x_1 + 5x_2 \leqslant 30 \\ 4x_1 + 2x_2 \leqslant 20 \\ x_1,\ x_2 \geqslant 0 \end{aligned}$$

Так как это задача максимизации, рассмотрим ее как уже дуальную. Соответственно, первые два ограничения превратятся в $p_i \geqslant 0$, а вторые два — в $A_j' p \geqslant c_j$. Здесь мы применили правила преобразования «в обратную» сторону.

$$\begin{aligned} \min_{p_1,p_2} 30p_1 + 20p_2 \\ 2p_1 + 4p_2 \geqslant 3 \\ 5p_1 + 2p_2 \geqslant 4 \\ p_1,\ p_2 \geqslant 0 \end{aligned}$$

1.3.2. Задача максимизации: вариант 2

Вы можете сначала преобразовать вашу задачу в задачу минимизации, а затем построить дуальную к ней.

Вернемся к задаче планирования выпуска

$$\begin{aligned} \max_{x_1,x_2} 3x_1 + 4x_2 \\ 2x_1 + 5x_2 \leqslant 30 \\ 4x_1 + 2x_2 \leqslant 20 \\ x_1,\ x_2 \geqslant 0 \end{aligned}$$

Преобразуем ее в задачу минимизации

$$\begin{aligned} \min_{x_1,x_2} -3x_1 - 4x_2 \\ 2x_1 + 5x_2 \leqslant 30 \\ 4x_1 + 2x_2 \leqslant 20 \\ x_1,\ x_2 \geqslant 0 \end{aligned}$$

В результате получим дуальную задачу (привила применяются в «прямую» сторону)

$$\begin{aligned} \max_{x_1,x_2} 30p_1 + 20p_2 \\ 2p_1 + 4p_2 \leqslant -3 \\ 5p_1 + 2p_2 \leqslant -4 \\ p_1,\ p_2 \leqslant 0 \end{aligned}$$

Сделав замену $\tilde{p}_i = -p_i$ и домножив, где нужно, на -1 получим

$$\begin{aligned} \min_{x_1,x_2} 30\tilde{p}_1 + 20\tilde{p}_2 \\ 2\tilde{p}_1 + 4\tilde{p}_2 \geqslant 3 \\ 5\tilde{p}_1 + 2\tilde{p}_2 \geqslant 4 \\ \tilde{p}_1,\ \tilde{p}_2 \geqslant 0 \end{aligned}$$

1.3.3. Теоремы дуальности

Слабая дуальность: если x и p — допустимые решения прямой и дуальной задач, то

$$b'p \leqslant c'x$$

Сильная дуальность: если x и p — оптимальные решения прямой и дуальной задач, то

$$b'p = c'x$$

Комплиментарность остатков: допустимые решения x и p являются оптимальными тогда и только тогда, когда

$$\begin{aligned} p_i(a_i'x - b_i) = 0 \quad \forall i \\ x_j(A_j'p - c_j) = 0 \quad \forall j \end{aligned}$$

Если $\exists i$ такое, что $a_i'x - b_i < 0$, тогда (в случае с примером с заводом) предприятию в оптимуме нужно меньше ресурса, чем у него есть. То есть его «теневая» цена p_i будет равна 0, он для предприятия «бесплатен».

Если $\exists j$ такое, что $A_j'p > c_j$, то стоимость всех затраченных на производство одной единицы товара j ресурсов больше его цены. Поэтому выгоднее перевести ресурсы на другие товары, следовательно $x_j = 0$.

1.4. Эксперименты: Задача с заводом

Вернемся к задаче планирования выпуска

$$\begin{aligned} \max_{x_1,x_2} 3x_1 + 4x_2 \\ 2x_1 + 5x_2 \leqslant 30 \\ 4x_1 + 2x_2 \leqslant 20 \\ x_1,\ x_2 \geqslant 0 \end{aligned}$$

Насладимся красивой картинкой.

[2]:
```
fig, (ax) = plt.subplots()

# Задаем ограничения по осям
ax.set_xlim(0, 15)
ax.set_ylim(0, 10)
# Генерируем значения `x`
x1 = np.linspace(0, 15)
```

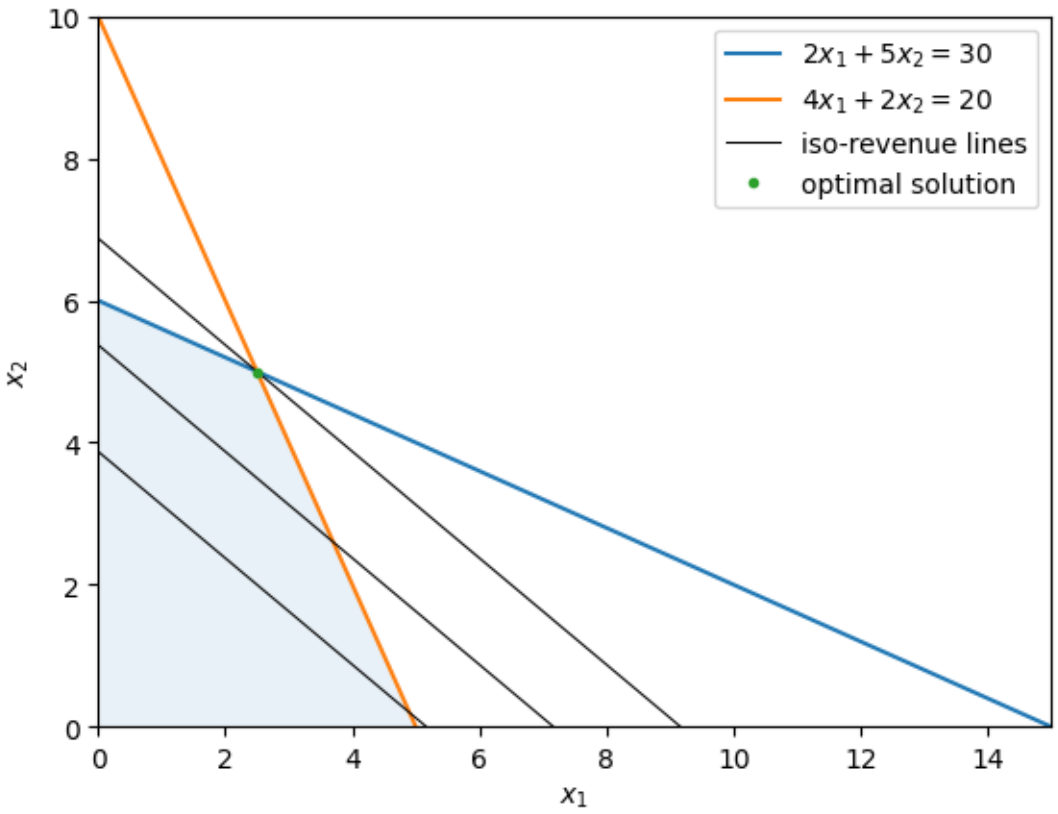
```
# Рисуем линии ограничений
ax.plot(x1, 6 - 0.4 * x1, label="$2x_1 + 5x_2=30$")
ax.plot(x1, 10 - 2 * x1, label="$4x_1 + 2x_2=20$")

# Рисуем допустимое множество
feasible_set = Polygon(np.array([[0, 0], [0, 6], [2.5, 5],␣
  ↪[5, 0]]), alpha=0.1)
ax.add_patch(feasible_set)

# Рисуем целевую функцию --- линии одинаковой прибыли
ax.plot(x1, 3.875 - 0.75 * x1, label="iso-revenue lines",␣
  ↪color="k", linewidth=0.75)
ax.plot(x1, 5.375 - 0.75 * x1, color="k", linewidth=0.75)
ax.plot(x1, 6.875 - 0.75 * x1, color="k", linewidth=0.75)

# Отобразим оптимальное решение
ax.plot(2.5, 5, ".", label="optimal solution")
ax.set_xlabel("$x_1$")
ax.set_ylabel("$x_2$")
ax.legend()

plt.show()
```



Определим объекты с параметрами задачи:

- матрица A
- векторы c и b

[3]:
```
c = np.array([3, 4])
A = np.array([[2, 5], [4, 2]])
b = np.array([30, 20])
```

1.4.1. `linprog`

Решим задачу при помощи **`scipy.optimize.linprog`**. Так как данная функция **минимизирует** целевую функцию, перед **`c`** надо поставить знак минус. **`linprog`** решает следующую задачу

$$\min_x c'x$$
$$A_{ub}x \leqslant b_{ub}$$
$$A_{eq}x = b_{eq}$$
$$lb \leqslant x \leqslant ub$$

A_{eq} и b_{eq} в нашем случае, очевидно, не определены.

[4]:
```
res = linprog(-c, A_ub=A, b_ub=b)
res
```

[4]:
```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -27.5
              x: [ 2.500e+00  5.000e+00]
            nit: 2
          lower:  residual: [ 2.500e+00  5.000e+00]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-6.250e-01 -4.375e-01]
  mip_node_count: 0
  mip_dual_bound: 0.0
         mip_gap: 0.0
```

Значение целевой функции равно 27.5 (не забываем, что мы ее домножили на -1).

Решим дуальную задачу (помните о том, какая задача является основной)

$$\min_{p_1,p_2} 30p_1 + 20p_2$$
$$2p_1 + 4p_2 \geqslant 3$$
$$5p_1 + 2p_2 \geqslant 4$$
$$p_1,\ p_2 \geqslant 0$$

Или в форме, более приближенной к стандартной

$$\begin{aligned} \min_{p_1, p_2} \; & 30p_1 + 20p_2 \\ -2p_1 - 4p_2 &\leqslant -3 \\ -5p_1 - 2p_2 &\leqslant -4 \\ p_1,\; p_2 &\geqslant 0 \end{aligned}$$

[5]:
```
res = linprog(b, A_ub=-A.T, b_ub=-c)
res
```

[5]:
```
        message: Optimization terminated successfully.
      (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: 27.5
              x: [ 6.250e-01  4.375e-01]
            nit: 2
          lower:  residual: [ 6.250e-01  4.375e-01]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-2.500e+00 -5.000e+00]
mip_node_count: 0
mip_dual_bound: 0.0
       mip_gap: 0.0
```

Как мы можем видеть, в оптимуме мы имеем сильную дуальность.

1.4.2. `linprog_simplex`

Также решим задачу при помощи `quantecon.optimize.linprog_simplex.linprog_sim`
Обратите внимание, что в этом случае нам **не надо** ставить минус перед `c`, так как эта функция решает задачу **максимизации**

$$\begin{aligned} \max_{x} \; & c'x \\ A_{ub}x &\leqslant b_{ub} \\ A_{eq}x &= b_{eq} \\ x &\geqslant 0 \end{aligned}$$

[6]:
```
res = linprog_simplex(c, A_ub=A, b_ub=b)
res
```

[6]:
```
SimplexResult(x=array([2.5, 5. ]), lambd=array([0.625 , 0.
  4375]), fun=27.5,
success=True, status=0, num_iter=4)
```

1.4.3. ortools

Также воспользуемся пакетом **ortools**.

> OR-Tools — это пакет программного обеспечения с открытым исходным кодом для оптимизации, предназначенный для решения сложнейших мировых проблем в области маршрутизации транспортных средств, потоков, целочисленного и линейного программирования, а также программирования в ограничениях.

Решим нашу сложнейшую мировую задачу.

[7]:
```
# Создадим экземпляр решателя GLOP (Google Linear Optimization Package)
solver = pywraplp.Solver.CreateSolver("GLOP")
```

Создадим переменные x_1 и x_2, которые могут принимать значения от 0 до ∞.

[8]:
```
x1 = solver.NumVar(0, solver.infinity(), "x1")
x2 = solver.NumVar(0, solver.infinity(), "x2")
```

Добавим ограничения задачи.

[9]:
```
solver.Add(2 * x1 + 5 * x2 <= 30.0)
solver.Add(4 * x1 + 2 * x2 <= 20.0)
```

[9]:
```
<ortools.linear_solver.pywraplp.Constraint; proxy of <Swig Object of type
'operations_research::MPConstraint *' at 0x000001C5A6128300> >
```

Зададим целевую функцию.

[10]:
```
solver.Maximize(3 * x1 + 4 * x2)
```

Попробуем решить нашу задачу. **ortools** позволяет проверить успешность решения задачи до погружения в результаты, что очень удобно.

[11]:
```
# Solve the system.
status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:
    print("Оптимум функции =", solver.Objective().Value())
    print(f"(x1, x2): ({x1.solution_value():.2}, {x2.solution_value():.2})")
else:
    print("Задача не имеет оптимального решения.")
```

```
Оптимум функции = 27.5
(x1, x2): (2.5, 5.0)
```

1.4.4. Z3

Можно извратиться и воспользоваться Z3. Это довольно странная программа, использующая LISP для внутренней работы!

[12]:
```
x1 = z3.Real("x1")
x2 = z3.Real("x2")
m = z3.Real("m")

o = z3.Optimize()

o.add(2 * x1 + 5 * x2 <= 30)
o.add(4 * x1 + 2 * x2 <= 20)
o.add(x1 >= 0, x2 >= 0)
o.add(3 * x1 + 4 * x2 == m)

h = o.maximize(m)

if o.check() == z3.sat:
    print(o.lower(h))
    print(o.model())
```

```
55/2
[m = 55/2, x2 = 5, x1 = 5/2]
```

Так как Z3 живет своей собственной увлекательной жизнью, то приходится извращаться до конца: чтобы вытащить значение в реальный Пайтон надо переводить результаты в текстовое представление.

[13]:
```
str(o.lower(h)), str(o.model()[x1])
```

[13]:
```
('55/2', '5/2')
```

Так как `float` не поймет число выше, надо либо использовать `eval`, что плохо, либо воспользоваться методом из z3.

[14]:
```
float(o.lower(h).as_decimal(prec=2)), float(o.model()[x1].
  ↪as_decimal(prec=2))
```

[14]:
```
(27.5, 2.5)
```

1.4.5. sympy

[15]:
```
x, y = map(Symbol, ["x", "y"])

restrictions = [
    2 * x + 5 * y <= 30,
    4 * x + 2 * y <= 20,
    x >= 0,
    y >= 0,
]
```

```
lpmax(3 * x + 4 * y, restrictions)
```

[15]: (55/2, {x: 5/2, y: 5})

1.5. Эксперименты: Задача с инвестиционным фондом

Вернемся к задаче с инвестиционным фондом

$$\begin{aligned}
\max_x \; & 1.30 \times 3x_1 + 1.06x_4 + 1.30x_5 \\
x_1 + x_2 &= 100000 \\
x_1 - 1.06x_2 + x_3 + x5 &= 0 \\
x_1 - 1.06x_3 + x_4 &= 0 \\
x_2 &\geqslant -20000 \\
x_3 &\geqslant -20000 \\
x_4 &\geqslant -20000 \\
x_5 &\leqslant 50000 \\
x_1,\, x_5 &\geqslant 0
\end{aligned}$$

Определим все необходимые матрицы и векторы.

[16]:
```
c = np.array([1.3 * 3, 0, 0, 1.06, 1.3])

b_eq = np.array([100_000, 0, 0])
A_eq = np.array(
    [
        [1, 1, 0, 0, 0],
        [1, -1.06, 1, 0, 1],
        [1, 0, -1.06, 1, 0],
    ]
)

b_ub = np.array([20_000, 20_000, 20_000, 50_000])
A_ub = np.array(
    [
        [0, -1, 0, 0, 0],
        [0, 0, -1, 0, 0],
        [0, 0, 0, -1, 0],
        [0, 0, 0, 0, 1],
    ]
)

bounds = [
    (0, None),
    (None, None),
    (None, None),
    (None, None),
    (0, None),
```

```
]

bounds_alt = [
    (0, None),
    (-20_000, None),
    (-20_000, None),
    (-20_000, None),
    (0, 50_000),
]
```

1.5.1. linprog

[17]:
```
res = linprog(-c, A_eq=A_eq, b_eq=b_eq, A_ub=A_ub, b_ub=b_ub, bounds=bounds)
res
```

[17]:
```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -141018.24349792697
              x: [ 2.493e+04  7.507e+04  4.649e+03 -2.000e+04  5.000e+04]
            nit: 0
          lower:  residual: [ 2.493e+04        inf        inf        inf
                              5.000e+04]
                marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                             0.000e+00]
          upper:  residual: [       inf        inf        inf        inf
                                    inf]
                marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                             0.000e+00]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00]
                marginals: [-1.376e+00 -1.299e+00 -1.225e+00]
        ineqlin:  residual: [ 9.507e+04  2.465e+04  0.000e+00  0.000e+00]
                marginals: [-0.000e+00 -0.000e+00 -1.650e-01 -1.470e-03]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

[18]:
```
res = linprog(-c, A_eq=A_eq, b_eq=b_eq, bounds=bounds_alt)
res
```

[18]:
```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -141018.24349792697
              x: [ 2.493e+04  7.507e+04  4.649e+03 -2.000e+04  5.000e+04]
            nit: 0
          lower:  residual: [ 2.493e+04  9.507e+04  2.465e+04  0.000e+00
                              5.000e+04]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  1.650e-01
                              0.000e+00]
          upper:  residual: [       inf        inf        inf        inf
                              0.000e+00]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                             -1.470e-03]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00]
                 marginals: [-1.376e+00 -1.299e+00 -1.225e+00]
        ineqlin:  residual: []
                 marginals: []
mip_node_count: 0
mip_dual_bound: 0.0
       mip_gap: 0.0
```

1.5.2. `linprog_simplex`

Для того, чтобы применить `linprog_simplex`, надо изменить определенным образом матрицы и векторы ограничений согласно

$$\tilde{x}_{2,3,4} = x_{2,3,4} + 20000$$

[19]:
```
b_eq2 = np.array([100_000, -1.06 * 20_000, -1.06 * 20_000])
A_eq2 = np.array(
    [
        [1, 1, 0, 0, 0],
        [1, -1.06, 1, 0, 1],
        [1, 0, -1.06, 1, 0],
    ]
)

b_ub2 = np.array([50_000])
```

```
A_ub2 = np.array([[0, 0, 0, 0, 1]])

res = linprog_simplex(c, A_ub=A_ub2, b_ub=b_ub2,␣
  ↪A_eq=A_eq2, b_eq=b_eq2)
res
```

[19]:
```
SimplexResult(x=array([ 4927.75474306, 95072.24525694,␣
  ↪24648.8252293 ,     0.
,
       50000.        ]), lambd=array([0.00147003, 1.
  ↪37644176, 1.29852997,
1.22502827]), fun=84218.24349792692, success=True,␣
  ↪status=0, num_iter=7)
```

Не забудем вычесть 20000 (и скорректировать значение функции)!

1.5.3. ortools

[20]:
```
solver = pywraplp.Solver.CreateSolver("GLOP")
x1 = solver.NumVar(0, solver.infinity(), "x1")
x2 = solver.NumVar(-20_000, solver.infinity(), "x2")
x3 = solver.NumVar(-20_000, solver.infinity(), "x3")
x4 = solver.NumVar(-20_000, solver.infinity(), "x4")
x5 = solver.NumVar(0, 50_000, "x5")

solver.Add(x1 + x2 == 100_000)
solver.Add(x1 - 1.06 * x2 + x3 + x5 == 0)
solver.Add(x1 - 1.06 * x3 + x4 == 0)

solver.Maximize(1.3 * 3 * x1 + 1.06 * x4 + 1.3 * x5)

status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:
    print("Оптимум 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()}")
else:
    print("Задача не имеет оптимального решения.")
```

```
Оптимум функции = 141018.24349792692
x1: 24927.754743058176
x2: 75072.24525694182
x3: 4648.825229300164
x4: -20000.0
x5: 50000.0
```

1.5.4. Z3

[21]:
```
z3.set_option(precision=10)
x1 = z3.Real("x1")
x2 = z3.Real("x2")
x3 = z3.Real("x3")
x4 = z3.Real("x4")
x5 = z3.Real("x5")
m = z3.Real("m")
o = z3.Optimize()
o.add(x1 + x2 == 100_000)
o.add(x1 - 1.06 * x2 + x3 + x5 == 0)
o.add(x1 - 1.06 * x3 + x4 == 0)
o.add(x1 >= 0, x5 >= 0, x5 <= 50_000)
o.add(x2 >= -20_000, x3 >= -20_000, x4 >= -20_000)
o.add(1.3 * 3 * x1 + 1.06 * x4 + 1.3 * x5 == m)
h = o.maximize(m)
if o.check() == z3.sat:
    for v in (m := o.model()):
        print(f"{str(v):>2} = {eval(str(m[v]))}")
```

```
x4 = -20000
 m = 141018.2434979269
x2 = 75072.24525694182
x3 = 4648.825229300163
x5 = 50000
x1 = 24927.754743058173
```

1.5.5. sympy

Обратите внимание на то, что далее в первых трех условиях мы использовали не `==`, а класс `Eq`. Дело в том, что `==`, согласно справке пакетв `sympy`, используется для проверки **полного структурного** соответствия двух объектов. А `Eq` используется для **математического** сравнения.

Таким образом, для сложных условий нужно использовать

- `Eq` вместо `==`,
- `Ne` вместо $\neq$.

Для неравенств лучше использовать стандартные Python-операторы. Но если очень хочется, то можно использовать

- `Ge` вместо $\geq$,
- `Gt` вместо `>`,
- `Le` вместо $\leq$,
- `Lt` вместо `<`.

[22]:
```
x, y, z, v, w = map(Symbol, ["x", "y", "z", "v", "w"])

restrictions = [
    Eq(x + y, 100_000),
    Eq(x - 1.06 * y + z + w, 0),
```

```
    Eq(x - 1.06 * z + v, 0),
    x >= 0,
    w >= 0,
    w <= 50_000,
    y >= -20_000,
    z >= -20_000,
    v >= -20_000,
]

lpmax(1.3 * 3 * x + 1.06 * v + 1.3 * w, restrictions)
```

[22]:
```
(141018.243497927,
 {v: -20000,
  w: 50000,
  x: 24927.7547430582,
  y: 75072.2452569418,
  z: 4648.82522930016})
```

1.6. Эксперименты: Дуальная задача фонда

Решим дуальную задачу (помните о том, какая задача является основной)

$$\begin{aligned}
\min_{p_1,\dots,p_7} \quad & 100000p_1 - 20000p_4 - 20000p_5 - 20000p_6 + 50000p_7 \\
p_1 + p_2 + p_3 &\geqslant 3.9 \\
p_2 + p_7 &\geqslant 1.3 \\
p_1 - 1.06p_2 + p_4 &= 0 \\
p_2 - 1.06p_3 + p_5 &= 0 \\
p_3 + p_6 &= 1.06 \\
p_{1,2,3} &\text{ свободны} \\
p_{4,5,6} &\leqslant 0 \\
p_7 &\geqslant 0
\end{aligned}$$

Для вывода вспомним, что

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & -1.06 & 1 & 0 & 1 \\ 1 & 0 & -1.06 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 100000 \\ 0 \\ 0 \\ -20000 \\ -20000 \\ -20000 \\ 50000 \end{pmatrix}, \quad c = \begin{pmatrix} 3.9 & 0 & 0 & 1.06 & 1.3 \end{pmatrix}$$

[23]:
```
c_d = np.array([100_000, 0, 0, -20_000, -20_000, -20_000,
 ↪50_000])

A_eqd = np.array(
    [
        [1, -1.06, 0, 1, 0, 0, 0],
```

```
        [0, 1, -1.06, 0, 1, 0, 0],
        [0, 0, 1, 0, 0, 1, 0],
    ]
)
b_eqd = np.array([0, 0, 1.06])

A_ubd = np.array(
    [
        [-1, -1, -1, 0, 0, 0, 0],
        [0, -1, 0, 0, 0, 0, -1],
    ]
)
b_ubd = np.array([-3.9, -1.3])

bounds_d = [
    (None, None),
    (None, None),
    (None, None),
    (None, 0),
    (None, 0),
    (None, 0),
    (0, None),
]
```

1.6.1. linprog

[24]:
```
res = linprog(
    c_d,
    A_eq=A_eqd,
    b_eq=b_eqd,
    A_ub=A_ubd,
    b_ub=b_ubd,
    bounds=bounds_d,
)
res
```

[24]:
```
        message: Optimization terminated successfully.␣
↪(HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: 141018.24349792683
              x: [ 1.376e+00  1.299e+00  1.225e+00  0.
↪000e+00  0.000e+00
                   -1.650e-01  1.470e-03]
            nit: 0
          lower:  residual: [       inf        inf       ␣
↪inf        inf
                                    inf        inf  1.470e-03]
```

```
                marginals: [ 0.000e+00  0.000e+00  0.
↪000e+00  0.000e+00
                            0.000e+00  0.000e+00  0.000e+00]
        upper:  residual: [       inf        inf        
↪inf  0.000e+00
                            0.000e+00  1.650e-01        inf]
                marginals: [ 0.000e+00  0.000e+00  0.
↪000e+00 -9.507e+04
                           -2.465e+04  0.000e+00  0.000e+00]
        eqlin:  residual: [ 0.000e+00  0.000e+00  0.
↪000e+00]
              marginals: [ 7.507e+04  4.649e+03 -2.000e+04]
      ineqlin:  residual: [ 0.000e+00  0.000e+00]
                marginals: [-2.493e+04 -5.000e+04]
mip_node_count: 0
mip_dual_bound: 0.0
       mip_gap: 0.0
```

1.6.2. ortools

[25]:
```
solver = pywraplp.Solver.CreateSolver("GLOP")
p1 = solver.NumVar(-solver.infinity(), solver.infinity(),
↪"p1")
p2 = solver.NumVar(-solver.infinity(), solver.infinity(),
↪"p2")
p3 = solver.NumVar(-solver.infinity(), solver.infinity(),
↪"p3")
p4 = solver.NumVar(-solver.infinity(), 0, "p4")
p5 = solver.NumVar(-solver.infinity(), 0, "p5")
p6 = solver.NumVar(-solver.infinity(), 0, "p6")
p7 = solver.NumVar(0, solver.infinity(), "p7")

solver.Add(p1 + p2 + p3 >= 3.9)
solver.Add(p2 + p7 >= 1.3)
solver.Add(p1 - 1.06 * p2 + p4 == 0)
solver.Add(p2 - 1.06 * p3 + p5 == 0)
solver.Add(p3 + p6 == 1.06)

solver.Minimize(100_000 * p1 - 20_000 * p4 - 20_000 * p5 -
↪20_000 * p6 + 50_000 * p7)

status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:
    print("Оптимум функции =", solver.Objective().Value())
    print(f"p1: {p1.solution_value()}")
    print(f"p2: {p2.solution_value()}")
    print(f"p3: {p3.solution_value()}")
    print(f"p4: {p4.solution_value()}")
```

```
    print(f"p5: {p5.solution_value()}")
    print(f"p6: {p6.solution_value()}")
    print(f"p7: {p7.solution_value()}")
else:
    print("Задача не имеет оптимального решения.")
```

```
Оптимум функции = 141018.24349792695
p1: 1.3764417640407092
p2: 1.29852996607614
p3: 1.225028269883151
p4: 0.0
p5: 0.0
p6: -0.1650282698831509
p7: 0.0014700339238598037
```