

Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Информационные системы и технологии

Линейное программирование
Отчёт по лабораторной работе №1

Работу выполнили:

Фирсова Д. А., студентка М33081
Деревицкая П. К., студентка М33081
Борздун А. В., студентка М33051

Преподаватель:

Свинцов М. В.

Санкт-Петербург,
2023

Оглавление

Структура JSON4

Симплекс метод5

Реализация симплекс-методаОшибка! Закладка не определена.

Цель работы: знакомство с линейным программированием

Задачи, решаемые во время выполнения работы:

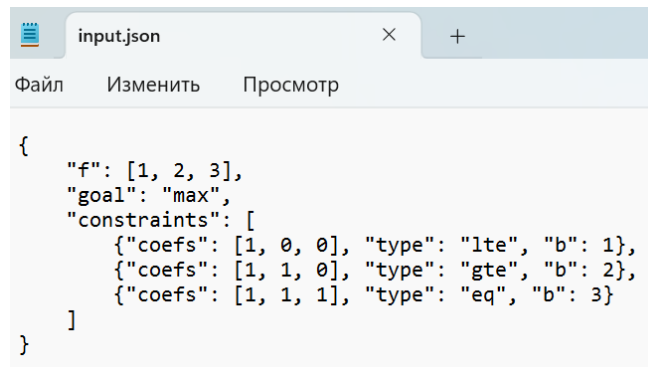
1. Реализация возможности ввода данных из файла в формате JSON.
2. Добавление балансирующих переменных для перехода от общей постановки к канонической форме задачи линейного программирования.
3. Реализация симплекс-метода для решения задачи.

Структура JSON

Ввод данных из файла

JSON — это формат данных, представляющий собой текстовое представление объектов. В JSON данные организованы в парах ключ-значение, где ключи — это строки, а значения могут быть строками, числами, логическими значениями, массивами, объектами, null.

У нас есть данные, которые находятся в файле input.json:



```
{
  "f": [1, 2, 3],
  "goal": "max",
  "constraints": [
    {"coefs": [1, 0, 0], "type": "lte", "b": 1},
    {"coefs": [1, 1, 0], "type": "gte", "b": 2},
    {"coefs": [1, 1, 1], "type": "eq", "b": 3}
  ]
}
```

Реализуем возможность ввода из этого файла:

```
import json

with open('input.json', 'r') as file:
    data = json.load(file)

coefficients = data["f"]
goal = data["goal"]
constraints = data["constraints"]
```

Выведем результат, чтобы сравнить его с оригинальной функцией:

```
print("Целевая функция:")
print("f(x) =", " + ".join([f"{coefficients[i]} · x{i + 1}"
                             for i in range(len(coefficients))]), f"→ {goal}")

print("Ограничения:")
for constraint in constraints:
    coefs = constraint["coefs"]
    constraint_type = constraint["type"]
    b = constraint["b"]
    constraint_sign = get_constraint_sign(constraint_type)
    constraint_str = " + ".join([f"{coefs[i]} · x{i + 1}"
                                   for i in range(len(coefs))])
    print(f"{constraint_str} {constraint_sign} {b}")
```

Целевая функция:
 $f(x) = 1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \rightarrow \max$
Ограничения:
 $1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 \leq 1$
 $1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 \geq 2$
 $1 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 = 3$

Функция сохраняется верно.

КЗЛП и СЗЛП

Теория

СЗЛП (система линейных уравнений и неравенств) и **КЗЛП** (каноническая форма линейного программирования) — это способы представления линейных программ в форме уравнений и неравенств, которые могут быть решены с использованием метода симплекс-метода.

Рассмотрим СЗЛП. Система линейных уравнений и неравенств представляет собой систему уравнений вида:

$$\begin{aligned} Ax &= b, \\ x &\geq 0, \end{aligned}$$

где A - матрица коэффициентов системы уравнений, x - вектор переменных, b - вектор свободных членов, и все переменные x_i должны быть неотрицательным.

Каноническая форма линейного программирования — это частный случай СЗЛП, где все ограничения являются уравнениями. В канонической форме отсутствуют неравенства. Если в исходной задаче присутствовали неравенства, они должны быть преобразованы в уравнения с использованием балансирующих переменных.

Пример:

Переведем к КЗЛП:

$$\begin{aligned} f(x) &= 1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \rightarrow \max \\ \begin{cases} x_1 \leq 1 \\ x_1 + x_2 \geq 2 \\ x_1 + x_2 + x_3 = 3 \end{cases} \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{aligned}$$

Вводим балансирующие переменные x_4 и x_5 , и получим:

$$\begin{aligned} x_1 + x_4 &= 1 \\ x_1 + x_2 - x_5 &= 2 \\ x_1 + x_2 + x_3 &= 3 \end{aligned}$$

Переведем к СЗЛП. Расширенная матрица системы ограничений-равенств данной задачи:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & -1 & 2 \\ 1 & 1 & 1 & 0 & 0 & 3 \end{bmatrix}$$

Приведем ее к единичной матрице:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ -1 & -1 & 0 & 0 & 1 & -2 \\ 1 & 1 & 1 & 0 & 0 & 3 \end{bmatrix}$$

В качестве базисных переменных принимаем $X = (3, 4, 5)$. Получаем:

$$x_1 + x_4 = 1$$

$$-x_1 - x_2 + x_5 = -2$$

$$x_1 + x_2 + x_3 = 3$$

Выразим базисные переменные через остальные:

$$x_4 = -x_1 + 1$$

$$x_5 = x_1 + x_2 - 2$$

$$x_3 = -x_1 - x_2 + 3$$

Подставим их в целевую функцию:

$$F(X) = x_1 + 2x_2 + 3(-x_1 - x_2 + 3)$$

или

$$F(X) = -2x_1 - x_2 + 9 \rightarrow \max$$

Симплекс метод

Теория

Симплекс-метод – алгоритмический подход к решению задач линейного программирования. Цель состоит в определении оптимальных значений переменных в линейной функции (называемой целевой функцией) при условии выполнения набора линейных ограничений.

Базовые понятия:

- *Опорный план*: начальное решение задачи линейного программирования, удовлетворяющее всем ограничениям.
- *Базисные и небазисные переменные*:
Базисные переменные – переменные, которые имеют ненулевые значения в опорном плане. Ненулевые значения базисных переменных связываются с ограничениями. Небазисные переменные – переменные, которые имеют нулевые значения в опорном плане.
- *Целевая функция и коэффициенты замещения*:
Целевая функция — это математическое выражение, которое нужно минимизировать или максимизировать. Целевая функция в опорном плане выражается через базисные переменные:

$$Q(x) = c_B^T x_B + c_N^T x_N = c_B^T x_B,$$

где (c_B^T) - вектор коэффициентов целевой функции для базисных переменных, и $(x_N = 0)$ в опорном плане.

Коэффициенты замещения (или свободные члены) вычисляются в симплекс-методе для небазисных переменных. Они показывают, на сколько увеличится (или уменьшится) значение целевой функции, если соответствующая небазисная переменная увеличится на единицу. Коэффициенты замещения вычисляются как:

$$z_j = c_j - \sum_{i=1}^m a_{ij} y_{B_i},$$

(z_j) - коэффициент замещения для переменной (x_j) ,
 (c_j) - коэффициент в целевой функции для переменной (x_j) ,
 (a_{ij}) - элемент матрицы (A) для (i) -й строки и (j) -го столбца,
 (y_{B_i}) - значения базисных переменных в опорном плане.

- *Выбор вводящей и исходящей переменных*:
Вводящая переменная — это небазисная переменная с отрицательным коэффициентом в коэффициентах замещения. Выбирается вводящая переменная для включения в опорный план с целью улучшения текущего значения целевой функции. Для того, чтобы найти вводящую переменную, нужно найти отрицательный коэффициент замещения $(z_j < 0)$ для небазисной переменной (x_j) .
Исходящая переменная — это базисная переменная, которая покидает опорный план. Выбирается исходящая переменная для замены ее новой вводящей переменной, чтобы достичь нового опорного плана с более низким значением целевой функции. Для того, чтобы найти исходящую переменную, нужно сначала вычислить отношения $(\theta_i = \frac{y_{B_i}}{a_{ij}})$ для всех $(a_{ij} > 0)$, связанных с выбранной вводящей переменной. После чего найти минимальное положительное (θ_i) . Эта переменная будет являться исходящей.
- *Оптимальное решение* – такое решение, при котором все коэффициенты замещения для небазисных переменных положительны или равны нулю, текущий опорный план является оптимальным, и значения базисных переменных представляют оптимальное решение задачи линейного программирования.

Математическое описание:

Рассмотрим задачу линейного программирования:

$$Q(\bar{x}) = c_1 x_1 + c_2 x_2 + \dots + c_n x_n \rightarrow \min$$

с системой ограничений следующего вида:

$$\begin{cases} a_{\{11\}} * x_1 + a_{\{12\}} * x_2 + \dots + a_{\{1n\}} * x_n = b_1 \\ \dots \\ a_{\{m,1\}} * x_1 + a_{\{m,2\}} * x_2 + \dots + a_{\{m,n\}} * x_n = b_m \end{cases}$$

Разрешим эту систему относительно переменных (x_1, x_2, \dots, x_m) :

$$\begin{cases} x_1 = a_{\{1,m+1\}} * x_{m+1} + \dots + a_{\{1,n\}} * x_n + b_1 \\ \dots \\ x_m = a_{\{m,m+1\}} * x_{m+1} + \dots + a_{\{m,n\}} * x_n + b_m \end{cases}$$

Здесь векторы условий $f(x) = (x_1, x_2, \dots, x_m)$ образуют базис. Переменные $(x_{\{m+1\}}, x_{\{m+2\}}, \dots, x_n)$ являются небазисными переменными. Целевую функцию можно выразить через небазисные переменные:

$$Q(f(x)) = c_{\{m+1\}} x_{\{m+1\}} + c_{\{m+2\}} x_{\{m+2\}} + \dots + c_n x_n + c \rightarrow \min.$$

Если приравнять небазисные переменные к нулю ($x_{\{m+1\}} = 0, x_{\{m+2\}} = 0, \dots, x_n = 0$), соответствующие базисные переменные примут значения $x_1 = b_1, x_2 = b_2, \dots, x_m = b_m$. Вектор \bar{x} с такими компонентами представляет собой угловую точку многогранника решений (допустимую) при условии, что $(b_i \geq 0)$ (опорный план).

Для нахождения оптимального решения задачи выбираются небазисная переменная и базисная переменная таким образом, чтобы после их обмена значение целевой функции уменьшилось. Повторение этой операции приведет к оптимальному решению задачи.

Таким образом, мы получили алгоритм решения. Запишем формально.

Алгоритмические шаги:

- 1) Инициализация:
 - i) Привести задачу линейного программирования к канонической форме.
 - ii) Выбрать начальное базисное решение и выразить основные переменные через неосновные.
 - iii) Рассчитать значения всех переменных в базисе.
- 2) Оценка оптимальности:
 - i) Вычислить коэффициенты целевой функции для неосновных переменных.
 - ii) Если все коэффициенты положительны (или в случае минимизации - отрицательны), текущее базисное решение оптимально. Завершить алгоритм.
- 3) Выбор входящей переменной:
 - i) Выбрать входящую переменную с отрицательным (положительным) коэффициентом в функции цели (в случае максимизации - наибольший положительный коэффициент, в случае минимизации - наибольший по модулю отрицательный коэффициент).
- 4) Выбор выходящей переменной:
 - i) Выбрать выходящую переменную путем нахождения минимального положительного отношения соответствующего правого члена к коэффициенту входящей переменной в соответствующем ограничении.

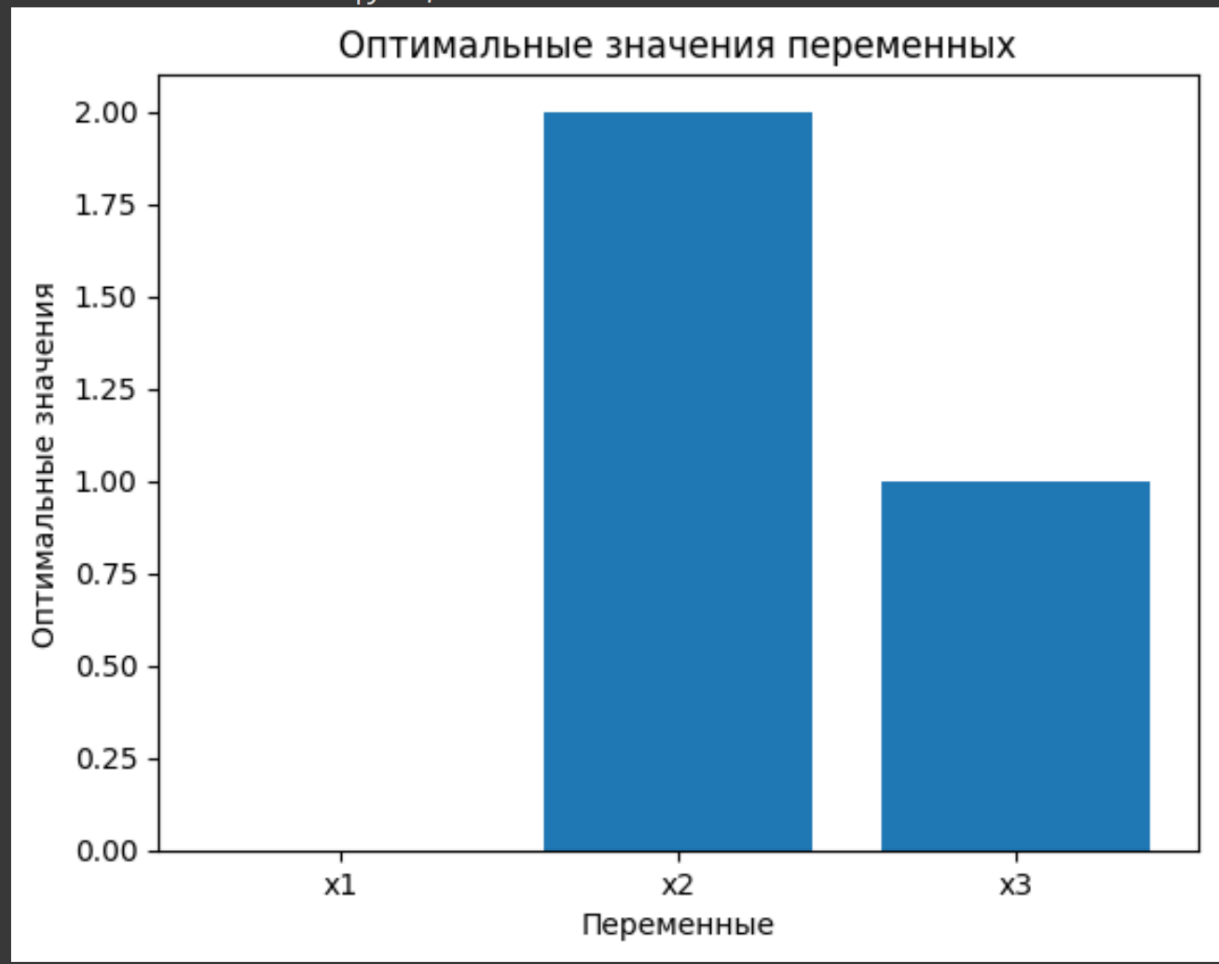
- ii) Пересчитать базис, выразив входящую переменную через выходящую переменную.
- 5) Обновление базисного решения:
 - i) Пересчитать значения всех переменных в базисе.
 - ii) Вернуться к шагу 2 для оценки оптимальности нового базисного решения.
- 6) Завершение:
 - i) Повторять шаги 3–5 до тех пор, пока не будет найдено оптимальное решение или будет обнаружено, что задача несовместна или неограничена.

Реализация симплекс-метода

Реализация данного метода представлена здесь в [Google Colab](#).

Получившийся ответ:

```
Оптимальные значения переменных: [0.0, 2.0, 1.0]  
Оптимальное значение функции: 7.0
```



Мы сравнили получившееся нами решение с решением, которое представлено во встроенной библиотеке Python - PuLP. Ответы получились одинаковые, значит наша реализация оказалась верной.

```

import pulp
import matplotlib.pyplot as plt
import numpy as np

# Определение задачи с использованием PuLP
coefficients = [1, 2, 3]
goal = pulp.LpMaximize # Цель - максимизация
lp_problem = pulp.LpProblem("Simplex_Method", goal)

# Определение переменных
x = [pulp.LpVariable(f"x{i}", lowBound=0) for i in range(1, len(coefficients) + 1)]

# Добавление целевой функции
lp_problem += pulp.lpDot(coefficients, x)

# Добавление ограничений
constraints = [
    {"coefs": [1, 0, 0], "type": "lte", "b": 1},
    {"coefs": [1, 1, 0], "type": "gte", "b": 2},
    {"coefs": [1, 1, 1], "type": "eq", "b": 3}
]

for constraint in constraints:
    if constraint["type"] == "lte":
        lp_problem += pulp.lpDot(constraint["coefs"], x) <= constraint["b"]
    elif constraint["type"] == "gte":
        lp_problem += pulp.lpDot(constraint["coefs"], x) >= constraint["b"]
    elif constraint["type"] == "eq":
        lp_problem += pulp.lpDot(constraint["coefs"], x) == constraint["b"]

# Решение задачи
lp_problem.solve()

# Вывод оптимальных значений переменных и значения целевой функции
optimal_values = [pulp.value(var) for var in x]
optimal_value = pulp.value(lp_problem.objective)
print("Оптимальные значения переменных:", optimal_values)
print("Оптимальное значение функции:", optimal_value)

# Создание графиков оптимальных значений переменных
variables = [f"x{i}" for i in range(1, len(coefficients) + 1)]
plt.bar(variables, optimal_values)
plt.xlabel("Переменные")
plt.ylabel("Оптимальные значения")
plt.title("Оптимальные значения переменных")
plt.show()

```

Выводы

В данной лабораторной работе был реализован симплекс-метод на языке программирования Python. Симплекс-метод является одним из основных алгоритмов решения задач линейного программирования.

Симплекс-метод представляет собой итерационный алгоритм, который позволяет находить оптимальное решение задачи линейного программирования. Он основывается на поиске опорных точек в многомерном пространстве, где функция цели достигает своего максимального или минимального значения при заданных ограничениях.

Основным преимуществом симплекс-метода является его эффективность и простота реализации. Он способен решать задачи линейного программирования с большим количеством переменных и ограничений. Также симплекс-метод позволяет находить оптимальное решение даже в случае, когда задача имеет неограниченное множество решений или не имеет решений вовсе.

Однако, симплекс-метод имеет и некоторые недостатки. Во-первых, он может быть неэффективным в случае, когда задача имеет большое количество ограничений и переменных. В таких случаях метод может требовать значительного количества итераций для нахождения оптимального решения. Во-вторых, симплекс-метод не гарантирует нахождение глобального оптимума, а только локального.

Симплекс-метод является важным инструментом при решении задач линейного программирования. Он широко применяется в различных областях, таких как экономика, производственное планирование, логистика и другие. Благодаря своей эффективности и простоте реализации, симплекс-метод позволяет находить оптимальные решения задач с ограничениями, что делает его полезным инструментом для анализа и оптимизации различных процессов и систем.