**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Отчет**

**по лабораторной работе № 1**

**Линейное программирование**

**Авторы:**

**Матевосян Игорь Вячеславович**

**Кичмарев Александр Вадимович**

**Назарян Михаил Аркадиевич**

**Факультет: ИТИП**

**Группа: М33081**

Предполагаемый язык выполнения лабораторных работ Python 3. Лабораторные работы выполняются студентами индивидуально или в группах по 2-3 человека. По результатам выполнения лабораторной работы необходимо подготовить

отчет. Отчет должен содержать описание реализованных вами алгоритмов, ссылку

на реализацию, необходимые тесты и таблицы.

Постановка задачи

1. Реализуйте возможность ввода данных из файла в формате JSON. Рекомендуемая структура JSON указана ниже.

2. При необходимости добавьте балансирующие переменные для перехода от об-

щей постановки к канонической форме задачи линейного программирования.

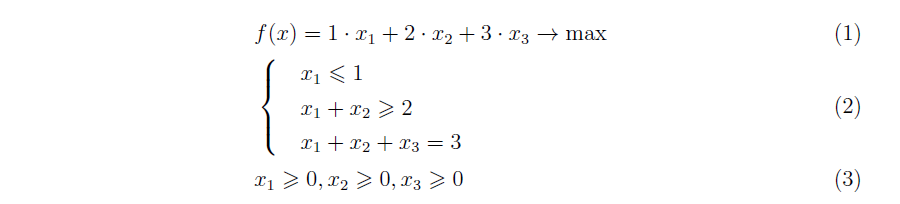
3. Реализуйте симплекс-метод для решения задачи.

4. Предусмотрите, что задача как может не иметь решений вообще, так и иметь

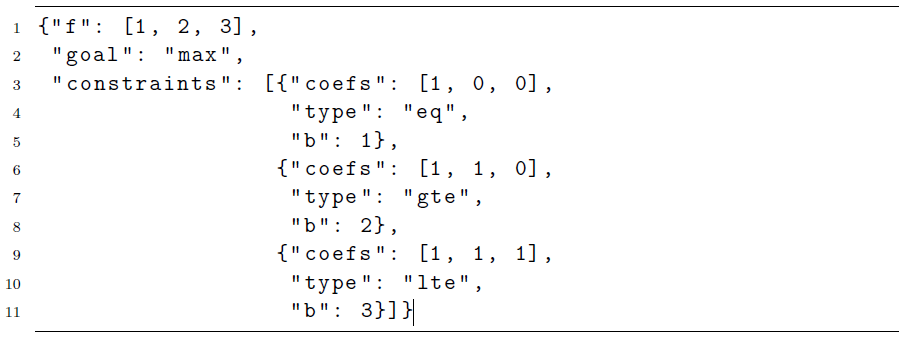
бесконечное количество решений.

Структура JSON

Задача линейного программирования (общая форма):



В формате JSON:



Решение:

алгоритм симплекс-метода:

from typing import Any  
  
  
def min\_check(seq):  
 x = 0  
 # убираем все `-1', потом ищем минимальный  
 while x < len(seq):  
 while x < len(seq) and seq[x] == '-1':  
 seq.pop(x)  
 x += 1  
 return min(seq)  
  
  
# a - коэффициенты левых частей неравенств  
# b - ограничения  
# c - коэффиценты функции  
# flip - "перевернуть", определяет задачу макс. или мин. True означает макс.  
# задача макс. преобразуется в задачу мин.  
# saveslack - сохранить ли дополнительные переменные, столбцы и прочее?  
def my\_simplex(a: list[float], b: list[float], c: list[Any], flip: bool, saveslack: bool, counter: int):  
 # преобразование задачи из макс. в мин.  
 if flip:  
 for x in range(0, len(c)):  
 c[x] = -c[x]  
 # базисы определяются значением индекса, на котором стоит единица  
 # иначе значение индекса -1  
 bases = []  
 # symbols - вспомогательный список для работы с преобразованием задачи в канонический вид  
 # из каждого уравнения берёт последний элемент, который может быть знаком неравенства  
 # если знак неравенства присутствует, то значение убирается для дальнейшего подсчёта  
 # иначе считается, что между уравнением и ограничением стоит равенство  
 symbols = []  
 for x in range(0, len(a)):  
 index = len(a[x]) - 1  
 if type(a[x][index]) is str:  
 symbols.append(a[x][index])  
 a[x].pop(index)  
 else:  
 symbols.append(a[x][index])  
 # сounter - переменная для удаления slack'ов  
 # все доп. переменные обычно находятся в конце, и их легко удалить  
 # counter - количество таких доп. переменных  
 # преобразование в канонич. вид  
 # если присутствует знак, то дальше идут действия в соответствии с ним  
 for x in range(0, len(a)):  
 # если знак присутствует, то в строке добавляется 1 (или -1), в остальных нули  
 if symbols[x] == '<':  
 c.append(0)  
 counter += 1  
 for y in range(0, len(a)):  
 if y != x:  
 a[y].append(0)  
 else:  
 a[y].append(1)  
 elif symbols[x] == '>':  
 c.append(0)  
 counter += 1  
 for y in range(0, len(a)):  
 if y != x:  
 a[y].append(0)  
 else:  
 a[y].append(-1)  
 # далее для определения, существуют ли везде базисы, работает этот цикл  
 # в первую очередь, мы определяем все уже существующие базисы  
 for x in range(0, len(a[0])):  
 # изначально мы предполагаем что столбец базисный, но с неопр. индексом (-1)  
 y = 0  
 baseIndex = -1  
 isBase = True  
 while isBase and y < len(b):  
 # если мы находим 1 или 0, то всё хорошо  
 isBase = a[y][x] == 0 or a[y][x] == 1  
 # однако нужно проверить, какой раз мы "напоролись" на единицу  
 # если уже не первый, то явно столбец тоже не базисный  
 if a[y][x] == 1:  
 if baseIndex > -1:  
 isBase = False  
 baseIndex = -1  
 else:  
 baseIndex = y  
 y += 1  
 # если всё хорошо, добавляем новый базисный индекс, иначе -1  
 if isBase:  
 bases.append(baseIndex)  
 else:  
 bases.append(-1)  
 baseExists = []  
 # далее проверяем, для всех ли строк существует базисный столбец  
 # для строки существует базисный столбец, если есть такой базисный столбец  
 # у которого на этой строке есть единица  
 for x in range(0, len(b)):  
 baseExists.append(False)  
 for x in range(0, len(a[0])):  
 if bases[x] > -1:  
 baseExists[bases[x]] = True  
 # если где-то нет базиса, добавляем  
 # если базиса для такой строки нет, то добавляем единицу в эту строку  
 # в остальные - нули  
 # в уравнение минимизации добавляем единицу  
 for x in range(0, len(b)):  
 if not baseExists[x]:  
 c.append(1)  
 counter += 1  
 for y in range(0, len(b)):  
 a[y].append(0)  
 a[x][len(a[0]) - 1] = 1  
 bases.append(x)  
 # запоминаем, какие столбцы у нас изначально были базисными  
 # понадобится для дальнейшей работы  
 bases\_prime = []  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 bases\_prime.append(x)  
 success = False  
 # подсчёт оценок  
 deltas = []  
 for y in range(0, len(a[0])):  
 d = 0  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 d += a[bases[x]][y] \* c[x]  
 d -= c[y]  
 deltas.append(d)  
 # если все оценки неположительные, то алгоритм заканчивает свою работу  
 success = True  
 for x in range(1, len(deltas)):  
 success = deltas[x] <= 0 and success  
 # пока есть положительные оценки:  
 while not success:  
 # ищем максимальную оценку  
 index = deltas.index(max(deltas))  
 # в столбце с макс. оценкой находим строку, у которой деление ограничения  
 # на соотв. значение из симпл. таблицы минимальное  
 # при этом не забываем, что делитель должен быть больше нуля  
 # иначе добавляем '-1'  
 potential = []  
 for y in range(0, len(b)):  
 if a[y][index] > 0:  
 potential.append(b[y] / a[y][index])  
 else:  
 potential.append('-1')  
 potential\_prime = []  
 # да, именно так, иначе питон не понимает, что я хочу создать дубликат  
 for x in range(0, len(potential)):  
 potential\_prime.append(potential[x])  
 # minp - минимальный элемент  
 minp = min\_check(potential)  
 # index2 - индекс скроки  
 index2 = potential\_prime.index(minp)  
 # смена базисных столбцов  
 # заменяемый базисный столбец теперь получает индекс -1  
 # а новый получает индекс строки  
 index3 = bases.index(index2)  
 bases[index] = index2  
 bases[index3] = -1  
 # теперь мы должны пересчитать симплекс таблицу  
 # сначала делим строку на элемент, чтобы получить один  
 # далее прибавляем эту строку, умноженную на минус элемент этого же столбца, но других строк  
 # к остальным строкам, чтобы получить нули  
 divider = a[index2][index]  
 for x in range(0, len(a[0])):  
 a[index2][x] /= divider  
 b[index2] /= divider  
 for y in range(0, len(a)):  
 if y != index2:  
 multiplier = -a[y][index]  
 for x in range(0, len(a[0])):  
 a[y][x] += a[index2][x] \* multiplier  
 b[y] += b[index2] \* multiplier  
 # пересчёт оценок  
 deltas = []  
 for y in range(0, len(a[0])):  
 d = 0  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 d += a[bases[x]][y] \* c[x]  
 d -= c[y]  
 deltas.append(d)  
 # проверка на окончание алгоритма  
 success = True  
 for x in range(0, len(deltas)):  
 success = deltas[x] <= 0 and success  
 # после окончания работы алгоритма переходим к непосредтсвенному выводу ответа  
 # result - значения переменных целевой функции  
 result = []  
 # убираем дополнительные базисы  
 bases = bases[:len(bases) - counter]  
 # если нужно сохранить остальное, то для этого используется параметр saveslack  
 if not saveslack:  
 for x in range(0, len(a)):  
 a[x] = a[x][:len(a[x]) - counter]  
 c = c[: len(c) - counter]  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 result.append(b[bases[x]])  
 else:  
 result.append(0)  
 # подсчёт значения целевой функции  
 f = 0  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 f += b[bases[x]] \* c[x]  
 if flip:  
 f = -f  
 return [a, b, bases\_prime, bases, f, result, c, counter]  
  
  
def my\_simplex\_double(a, b, c, flip, saveslack, counter):  
 # преобразование задачи из макс. в мин.  
 if flip:  
 for x in range(0, len(c)):  
 c[x] = -c[x]  
 # базисы определяются значением индекса, на котором стоит единица  
 # иначе значение индекса -1  
 bases = []  
 # symbols - вспомогательный список для работы с преобразованием задачи в канонический вид  
 # из каждого уравнения берёт последний элемент, который может быть знаком неравенства  
 # если знак неравенства присутствует, то значение убирается для дальнейшего подсчёта  
 # иначе считается, что между уравнением и ограничением стоит равенство  
 symbols = []  
 for x in range(0, len(a)):  
 index = len(a[x]) - 1  
 if type(a[x][index]) is str:  
 symbols.append(a[x][index])  
 a[x].pop(index)  
 else:  
 symbols.append(a[x][index])  
 # сounter - переменная для удаления slack'ов  
 # все доп. переменные обычно находятся в конце, и их легко удалить  
 # counter - количество таких доп. переменных  
 # преобразование в канонич. вид  
 # если присутствует знак, то дальше идут действия в соответствии с ним  
 for x in range(0, len(a)):  
 # если знак присутствует, то в строке добавляется 1 (или -1), в остальных нули  
 if symbols[x] == '<':  
 c.append(0)  
 counter += 1  
 for y in range(0, len(a)):  
 if y != x:  
 a[y].append(0)  
 else:  
 a[y].append(1)  
 elif symbols[x] == '>':  
 c.append(0)  
 counter += 1  
 for y in range(0, len(a)):  
 if y != x:  
 a[y].append(0)  
 else:  
 a[y].append(-1)  
 # далее для определения, существуют ли везде базисы, работает этот цикл  
 # в первую очередь, мы определяем все уже существующие базисы  
 for x in range(0, len(a[0])):  
 # изначально мы предполагаем что столбец базисный, но с неопр. индексом (-1)  
 y = 0  
 baseIndex = -1  
 isBase = True  
 while isBase and y < len(b):  
 # если мы находим 1 или 0, то всё хорошо  
 isBase = a[y][x] == 0 or a[y][x] == 1  
 # однако нужно проверить, какой раз мы "напоролись" на единицу  
 # если уже не первый, то явно столбец тоже не базисный  
 if a[y][x] == 1:  
 if baseIndex > -1:  
 isBase = False  
 baseIndex = -1  
 else:  
 baseIndex = y  
 y += 1  
 # если всё хорошо, добавляем новый базисный индекс, иначе -1  
 if isBase:  
 bases.append(baseIndex)  
 else:  
 bases.append(-1)  
 baseExists = []  
 # далее проверяем, для всех ли строк существует базисный столбец  
 # для строки существует базисный столбец, если есть такой базисный столбец  
 # у которого на этой строке есть единица  
 for x in range(0, len(b)):  
 baseExists.append(False)  
 for x in range(0, len(a[0])):  
 if bases[x] > -1:  
 baseExists[bases[x]] = True  
 # если где-то нет базиса, добавляем  
 # если базиса для такой строки нет, то добавляем единицу в эту строку  
 # в остальные - нули  
 # в уравнение минимизации добавляем единицу  
 for x in range(0, len(b)):  
 if not baseExists[x]:  
 c.append(1)  
 counter += 1  
 for y in range(0, len(b)):  
 a[y].append(0)  
 a[x][len(a[0]) - 1] = 1  
 bases.append(x)  
 # запоминаем, какие столбцы у нас изначально были базисными  
 # понадобится для дальнейшей работы  
 bases\_prime = []  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 bases\_prime.append(x)  
 # подсчёт оценок  
 deltas = []  
 for y in range(0, len(a[0])):  
 d = 0  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 d += a[bases[x]][y] \* c[x]  
 d -= c[y]  
 deltas.append(d)  
 # далее всё почти как в симплексном методе  
 # меняется принцип работы  
 # условие окончание теперь другое  
 # работа алгоритма завершается, если все ограничения неотрицательные  
 success = True  
 for x in range(0, len(b)):  
 success = b[x] >= 0 and success  
 while not success:  
 # ищется минимальное по значению ограничению  
 index = b.index(min(b))  
 # дальше ищется минимальное частное от значений  
 # оценок на соотв. значения строки  
 potential = []  
 for y in range(0, len(deltas)):  
 if a[index][y] != 0 and bases[y] == -1 and deltas[y] < 0:  
 potential.append(deltas[y] / a[index][y])  
 else:  
 potential.append('-1')  
 potential\_prime = []  
 for x in range(0, len(potential)):  
 potential\_prime.append(potential[x])  
 minp = min\_check(potential)  
 index2 = potential\_prime.index(minp)  
 # смена базисных столбцов  
 # заменяемый базисный столбец теперь получает индекс -1  
 # а новый получает индекс строки  
 index3 = bases.index(index)  
 bases[index2] = index  
 bases[index3] = -1  
 # теперь мы должны пересчитать симплекс таблицу  
 # сначала делим строку на элемент, чтобы получить один  
 # далее прибавляем эту строку, умноженную на минус элемент этого же столбца, но других строк  
 # к остальным строкам, чтобы получить нули  
 divider = a[index][index2]  
 for x in range(0, len(a[0])):  
 a[index][x] /= divider  
 b[index] /= divider  
 for y in range(0, len(a)):  
 if y != index:  
 multiplier = -a[y][index2]  
 for x in range(0, len(a[0])):  
 a[y][x] += a[index][x] \* multiplier  
 b[y] += b[index] \* multiplier  
 # пересчёт оценок  
 deltas = []  
 for y in range(0, len(a[0])):  
 d = 0  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 d += a[bases[x]][y] \* c[x]  
 d -= c[y]  
 deltas.append(d)  
 # проверка на окончание алгоритма  
 success = True  
 for x in range(0, len(b)):  
 success = b[x] >= 0 and success  
 # после окончания работы алгоритма переходим к непосредтсвенному выводу ответа  
 # result - значения переменных целевой функции  
 result = []  
 # убираем дополнительные базисы  
 bases = bases[:len(bases) - counter]  
 # если нужно сохранить остальное, то для этого используется параметр saveslack  
 if not saveslack:  
 for x in range(0, len(a)):  
 a[x] = a[x][:len(a[x]) - counter]  
 c = c[: len(c) - counter]  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 result.append(b[bases[x]])  
 else:  
 result.append(0)  
 # подсчёт значения целевой функции  
 f = 0  
 for x in range(0, len(bases)):  
 if bases[x] > -1:  
 f += b[bases[x]] \* c[x]  
 if flip:  
 f = -f  
 return [a, b, bases\_prime, bases, f, result, c, counter]

Парсинг json файла:

import json  
  
from typing import Any  
  
OP\_CAST = {'eq': '=', 'gte': '>', 'lte': '<'}  
  
  
def parse\_data\_json(file\_name: str) -> tuple[list[float], list[float], list[Any]]:  
 data = json.load(open(file\_name))  
 c = data['f']  
 b = [constraint['b'] for constraint in data['constraints']]  
  
 a = [constraint['coefs'] + list(OP\_CAST[constraint['type']])  
 for constraint in data['constraints']]  
 return a, b, c

**Первый тест(data.json):**

{  
 "f": [  
 3,  
 5,  
 4  
 ],  
 "goal": "max",  
 "constraints": [  
 {  
 "coefs": [  
 4,  
 5,  
 6  
 ],  
 "type": "lte",  
 "b": 6  
 },  
 {  
 "coefs": [  
 3,  
 6,  
 3  
 ],  
 "type": "lte",  
 "b": 9  
 },  
 {  
 "coefs": [  
 5,  
 2,  
 1  
 ],  
 "type": "lte",  
 "b": 3  
 },  
 {  
 "coefs": [  
 3,  
 4,  
 5  
 ],  
 "type": "lte",  
 "b": 5  
 },  
 {  
 "coefs": [  
 1,  
 2,  
 3  
 ],  
 "type": "lte",  
 "b": 4  
 }  
 ]  
}

**Результат:**

СИМПЛЕКС-МЕТОД

Исходные данные:

Таблица коэффицентов и знаков:

[4, 5, 6, '<']

[3, 6, 3, '<']

[5, 2, 1, '<']

[3, 4, 5, '<']

[1, 2, 3, '<']

Коэфиценты целевой функции:

[3, 5, 4]

Ограничения:

[6, 9, 3, 5, 4]

Результат работы симлпекс-метода:

Таблица коэффицентов:

[0.8, 1.0, 1.2, 0.2, 0.0, 0.0, 0.0, 0.0]

[-1.8000000000000007, 0.0, -4.199999999999999, -1.2000000000000002, 1.0, 0.0, 0.0, 0.0]

[3.4, 0.0, -1.4, -0.4, 0.0, 1.0, 0.0, 0.0]

[-0.20000000000000018, 0.0, 0.20000000000000018, -0.8, 0.0, 0.0, 1.0, 0.0]

[-0.6000000000000001, 0.0, 0.6000000000000001, -0.4, 0.0, 0.0, 0.0, 1.0]

Значение целевой функции:

6.0

Значения переменных целевой функции:

[0, 1.2, 0]

**Второй тест (data0.json):**

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

**Результат:**

СИМПЛЕКС-МЕТОД

Исходные данные:

Таблица коэффицентов и знаков:

[1, 0, 0, '=']

[1, 1, 0, '>']

[1, 1, 1, '<']

Коэфиценты целевой функции:

[1, 2, 3]

Ограничения:

[1, 2, 3]

Результат работы симлпекс-метода:

Таблица коэффицентов:

[1, 0, 0, 0, 0, 1, 0]

[1, 1, 0, -1, 0, 0, 1]

[1, 1, 1, 0, 1, 0, 0]

Значение целевой функции:

9

Значения переменных целевой функции:

[0, 0, 3]

**Третий тест (data2.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  
 },  
 {  
 "coefs": [  
 1,  
 0,  
 0  
 ],  
 "type": "gte",  
 "b": 0  
 },  
 {  
 "coefs": [  
 0,  
 1,  
 0  
 ],  
 "type": "gte",  
 "b": 0  
 },  
 {  
 "coefs": [  
 0,  
 0,  
 1  
 ],  
 "type": "gte",  
 "b": 0  
 }  
 ]  
}

**Результат:**

СИМПЛЕКС-МЕТОД

Исходные данные:

Таблица коэффицентов и знаков:

[1, 0, 0, '<']

[1, 1, 0, '>']

[1, 1, 1, '=']

[1, 0, 0, '>']

[0, 1, 0, '>']

[0, 0, 1, '>']

Коэфиценты целевой функции:

[1, 2, 3]

Ограничения:

[1, 2, 3, 0, 0, 0]

Результат работы симлпекс-метода:

Таблица коэффицентов:

[0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0]

[0.0, 0.0, 0.0, 0.0, -1.0, 1.0, 1.0, 0.0, 1.0, 0.0, -1.0, -1.0, 0.0]

[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, -1.0, 1.0, 0.0, 0.0, -1.0]

[1.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]

[0.0, 1.0, 0.0, 0.0, -1.0, 1.0, 0.0, 0.0, 1.0, 0.0, -1.0, 0.0, 0.0]

[0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, -1.0, 1.0, 0.0, 0.0, 0.0]

Значение целевой функции:

7.0

Значения переменных целевой функции:

[0.0, 2.0, 1.0]