Российский государственный университет нефти и газа (НИУ)

имени И.М. Губкина

ОТЧЕТ

о выполнении домашнего задания №3

по дисциплине «Методы оптимизации»

Выполнил: Коротченя И.С.

Проверила: Рыжова Л.Л.

Группа: АМ-20-06

Москва, 2023

**Постановка задачи**

Для исходных данных, вариант которых соответствует вашему номеру по списку, необходимо решить задачу линейного программирования вида:

1. Решение реализовать с помощью симплекс метода

**Исходные данные**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C = | (0 | 7 | -5 | 7 | -5) |
|  |  |  |  |  |  |
|  | 3 | 2 | 5 | 3 | 8 |
| A = | -6 | -5 | 0 | 6 | -4 |
|  | 7 | 3 | 3 | 9 | 5 |
|  |  |  |  |  |  |
| B = | (21 | -9 | 27 | ) |  |

**Решение**

Сперва введем искуссвенный базис . Получим систему, которую можно записать следующим образом:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | 3 | 2 | 5 | 3 | 8 | 21 |
|  | -6 | -5 | 0 | 6 | -4 | -9 |
|  | 7 | 3 | 3 | 9 | 5 | 27 |
|  | 0 | -7 | 5 | -7 | 5 | 0 |

Далее избавимся от отрицательных элементов в столбце b для этого домножим строки соответсвующие таким элементам на -1, получим:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | 3 | 2 | 5 | 3 | 8 | 21 |
|  | 6 | 5 | 0 | -6 | 4 | 9 |
|  | 7 | 3 | 3 | 9 | 5 | 27 |
|  | 0 | -7 | 5 | -7 | 5 | 0 |

Затем решим вспомогательную задачу вида:

Для того, выразить через небазисные переменные, прибавим все строки со знаком минус к , тогда получим:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | 3 | 2 | 5 | 3 | 8 | 21 |
|  | -6 | -5 | 0 | 6 | -4 | -9 |
|  | 7 | 3 | 3 | 9 | 5 | 27 |
|  | 0 | -7 | 5 | -7 | 5 | 0 |
|  | -16 | -10 | -8 | -6 | -17 | -57 |

Далее будем действовать с помощью симплекс метода, для того максимизировать целевую функцию путем вывода из базиса переменных . Алгоритм

1. В базис вводиться переменная для которой, коээфициент в строке минимальный и неположительный, стобец в котором находиться эта переменная - разрешающий
2. Из базиса выводиться переменная, для которой коэфицианты Котта () минимален и неотрицателен. Данная строка — разрешающая.
3. Разрешающий элемент находиться на пересечении разрешающий строки и разрешающего столбца, для него выполняем алгоритм смены базиса.
4. Удаляем из симплекс таблицы столбец, соответсвующий переменной y, которая была выведена из базиса.

Проделывая такую процедуру пока не избавимся от искусственных переменных получим:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | -0.179 | 0.174 | 0.995 |
|  | 0.096 | 0.66 | 1.755 |
|  | 0.59 | -0.266 | 1.324 |
|  | -8.732 | 2.918 | -1.814 |
|  | 0 | 0 | 0 |

Теперь, удалимм последнюю строку и будем решать задачу оптимизации с помощью симплекс метода для Сиплекс-таблицы следующего вида:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | -0.179 | 0.174 | 0.995 |
|  | 0.096 | 0.66 | 1.755 |
|  | 0.59 | -0.266 | 1.324 |
|  | -8.732 | 2.918 | -1.814 |

Дважды выполнив процедуру смены базиса получим симплекс-таблицу:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 0.325 | -0.132 | 1.192 |
|  | -0.231 | 1.423 | 2.192 |
|  | 1.59 | 0.641 | 3.231 |
|  | 14.556 | 1.444 | 20 |

**Программная реализация**

Для реализации данной задачи, создадим класс – Симплекс-таблица и реализуем в нем метод смены базиса, решения вспомогательной задачи и перехода к следующей симплекс таблицы.

При инициализации класса будем создавать двумерный массив, соответствующей текущей симплекс таблицы с помощью искусственного базиса, от которого будем избавляться с помощью метода реализующего решение вспомогательной задачи.

После чего будем будем вызывать метод реализующий переход к следующий симплекс таблице, пока он не начинет возвращать значение -1, которая символизирует что критерий оптимальности выполнен.

**Приложение**

import numpy as np  
  
class SymplexTable:  
 def \_\_init\_\_(self, A, b, c):  
 first\_line = [' '] + [f'-x{i}' for i in range(1, A.shape[1] + 1)] + ['b']  
 last\_line = ['L'] + [-i for i in c[0]] + [0]  
 self.table = [first\_line]  
 for i in range(A.shape[0]):  
 if b[i, 0] > 0:  
 new\_line = [f'y{i + 1}'] + [A[i, k] for k in range(A.shape[1])] + [b[i, 0]]  
 else:  
 new\_line = [f'y{i + 1}'] + [-A[i, k] for k in range(A.shape[1])] + [-b[i, 0]]  
 self.table.append(new\_line)  
  
 self.table.append(last\_line)  
 self.rows = len(self.table)  
 self.columns = len(self.table[0])  
 print('Start table')  
 self.print()  
  
 def change\_basic(self, row: int, column: int):  
 permissive\_elem = self.table[row][column]  
  
 self.table[0][column], self.table[row][0] = '-' + self.table[row][0], self.table[0][column].replace('-', '')  
  
 self.table[row][column] = 1  
 for i in range(1, self.rows):  
 if i == row:  
 continue  
 self.table[i][column] = -self.table[i][column]  
  
 for i in range(1, self.rows):  
 if i == row:  
 continue  
 for j in range(1, self.columns):  
 if j == column:  
 continue  
 self.table[i][j] = self.table[i][j] \* permissive\_elem + self.table[i][column] \* self.table[row][  
 j] # другой знак так как в актуальной  
  
 for i in range(1, self.rows):  
 for j in range(1, self.columns):  
 self.table[i][j] = self.table[i][j] / permissive\_elem  
  
 def print(self):  
 matrix = np.array(self.table)  
 col\_widths = [max([len(row[i]) if '.' not in row[i] else len(row[i].split('.')) + 6 for row in matrix]) for i in  
 range(len(matrix[0]))] # max в кажом столбце  
  
 res = ''  
 for row in matrix:  
 res += " | ".join(  
 row[i].ljust(col\_widths[i]) if '.' not in row[i] else str(round(float(row[i]), 3)).ljust(col\_widths[i])  
 for i in range(len(row))) + '\n' # выравниваем до макс  
 print(res)  
  
 def find\_permissive\_column(self):  
 minn = 0  
 permissive\_column = -1  
 for j in range(1, self.columns - 1):  
 if self.table[-1][j] <= minn:  
 minn = self.table[-1][j]  
 permissive\_column = j  
 return permissive\_column  
  
 def find\_permissive\_row(self, permissive\_column, y):  
 cote = []  
 for i in range(1, self.rows):  
 if 'L' not in self.table[i][0] and self.table[i][permissive\_column] > 0:  
 cote.append((self.table[i][-1]/self.table[i][permissive\_column], i))  
 cote.sort(key=lambda x: x[0])  
 if cote:  
 return cote[0][1]  
 return -1  
  
 def next\_symplex\_table(self):  
 permissive\_column = self.find\_permissive\_column()  
 if permissive\_column == -1:  
 print('Stop')  
 return -1  
 permissive\_row = self.find\_permissive\_row(permissive\_column, y=True)  
 if permissive\_row == -1:  
 print('Stop')  
 return -1  
 self.change\_basic(permissive\_row, permissive\_column)  
 print(f'after permessive elem ({permissive\_row + 1}, {permissive\_column + 1})')  
 self.print()  
  
 def support\_task(self):  
 L1 = []  
 for j in range(1, self.columns - 1):  
 temp = 0  
 for i in range(1, self.rows - 1):  
 temp += -self.table[i][j]  
 L1.append(temp)  
 self.table.append(['L1'] + [i for i in L1] + [sum([-self.table[i][-1] for i in range(1, self.rows)])])  
 self.rows += 1  
 print('table with L1')  
 self.print()  
  
 def make\_self\_basic(self):  
 self.support\_task()  
 while self.y\_in\_basic():  
 self.next\_symplex\_table()  
 self.delete\_y\_column()  
 if self.table[-1][-1] == 0:  
 print('Successfully brought to own basis')  
 else:  
 print('Something went wrong =(')  
 self.table.pop(-1)  
 self.rows -= 1  
 print('simplex table in self basic')  
 self.print()  
  
 def symplex\_method(self):  
 while True:  
 a = self.next\_symplex\_table()  
 if a == -1:  
 break  
  
  
 def delete\_y\_column(self):  
 for j in range(1, self.columns):  
 if 'y' in self.table[0][j]:  
 self.delete\_column(j)  
 break  
  
 def delete\_column(self, column\_id):  
 for i in range(self.rows):  
 self.table[i].pop(column\_id)  
 self.columns -= 1  
  
 def y\_in\_basic(self):  
 for i in range(1, self.rows):  
 if 'y' in self.table[i][0]:  
 return True  
 return False  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = np.array([  
 [3, 2, 5, 3, 8],  
 [-6, -5, 0, 6, -4],  
 [7, 3, 3, 9, 5]  
 ])  
 c = np.array([[0, 7, -5, 7, -5]])  
 b = np.array([[21], [-9], [27]])  
  
 table = SymplexTable(A, b, c)  
  
 table.make\_self\_basic()  
 table.symplex\_method()