Российский государственный университет нефти и газа (НИУ)

имени И.М. Губкина

ОТЧЕТ

о выполнении домашнего задания №2

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. Привести систему к базисному виду, если это необходимо, с помощью процедуры Жордана и искусственного базиса. Реализовать это программно, с помощью написания соответствующей процедуры.
2. Для таблицы Жордана в собственном базисе решить задачу геометрическим методом в координатах небазисных переменных, попутно отмечая на графике как опорные, так и остальные базисные решения, образующиеся пересечением прямых из ограничений и координатных осей.
3. Решить задачу линейного программирования методом полного перебора. Для этого необходимо:
4. Доказать ограниченность целевой функции;
5. Реализовать метод перебора;
6. «Идентифицировать» на графике геометрического метода все найденные базисные решения.

**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 докажем, что целевая функция действительно ограничена на множестве допустимых значений. Для этого запишем первое уравнение системы:

.

Так как , то наибольшее значение для каждого будет достигаться, когда все . Тогда получим:

Так как все переменные ограничены, то можно гарантировать, что функция цели также будет ограничена.

Далее необходимо привести систему к базисному виду, для этого воспользуемся искусственным базисом и алгоритмом Жордана.

В соответствие с методом искусственного базиса введем 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 с помощью преобразования Жордана будем по одной выводить из базиса искусственные переменные по следующему алгоритму:

1. Выбираем строку, в которой находиться искусственная переменная .
2. Выбираем столбец, в котором находиться один из .
3. Меняем местами и .
4. На пересечении разрешающей строки и столбца ставим 1
5. Элементы разрешающей строки переносим в новую таблицу без изменений
6. Элементы разрешающего столбца переносим в новую таблицу меняя знак
7. Все остальные элементы вычисляем по формуле прямоугольника.

Где a – разрешающий элемент, пересечение разрешающего столбца и строки, в которой находиться искомый элемент, пересечение разрешающего строки и столбца в которой находиться искомый элемент.

1. Все элементы новой таблицы делятся на разрешающий элемент
2. Удаляем столбец, в первой строке которого стоит (так как это искусственная переменная, равная 0).

Проделывая такую процедуру пока не избавимся от искусственных переменных получим:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 3.079 | -0.408 | 3.671 |
|  | -4.895 | 1.289 | -2.605 |
|  | 0.711 | 1.329 | 3.039 |
|  | -44.816 | 7.382 | -33.434 |

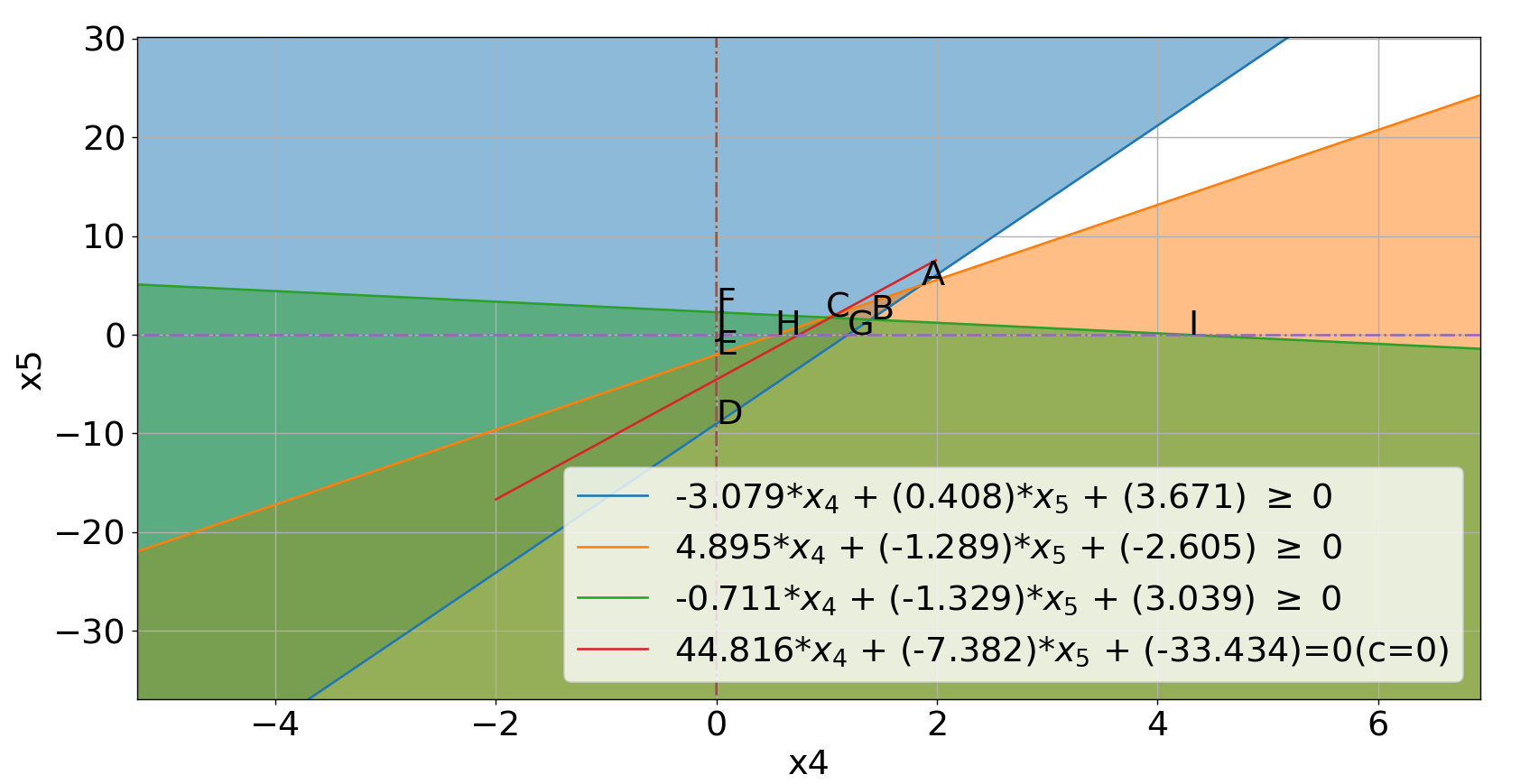
Далее необходимо найти оптимальное решение с помощью метода перебора, однако необходимо также построить геометрическое решение, что возможно так как количество небазисных переменных равно 2 (nm). Для построения график был выбран базис .

Для получения всех необходимых точек будем в процессе метода перебора запоминать значения переменных и ставить им в соответствие какую-либо точку. Переход между базисами будем осуществляется аналогично переходу от искусственного базиса, не учитывая 9 шаг.

В результате метода перебора было найдено 10 базисных решений:

|  |  |  |  |
| --- | --- | --- | --- |
| Небазисные переменные | Значение целевой функции | Опорность | Буква |
|  | 20 | Да | G |
|  | 17.77 | Да | B |
|  | -1.81 | Да | C |
|  | -9.58 | Да | H |
|  | 158.28 | Нет | I |
|  | 33 | Нет | D |
|  | 12.72 | Нет | A |
|  | -18.52 | Нет | E |
|  | 33.43 | Нет | J |
|  | 50.32 | Нет | F |

После получения базисных решений отметим их на графике. Области на графики получаются из условия не отрицательности базисных переменных (). Кроме того, построим целевую функцию (обозначена красным цветом). Тогда, исходя из уравнения целевой функции можно удостовериться в том, что опорное решений G будет искомым.



**Программная реализация**

Для реализации данной задачи, создадим класс – Симплекс-таблица и реализуем в нем метод смены базиса. При инициализации класса будем создавать двумерный массив, соответствующей текущей симплекс таблицы с помощью искусственного базиса, от которого сразу же будем избавляться с помощью метода смены базиса.

Добавим в класс метод, задача которого будет создавать график. График будем строить для базиса .

Добавим в класс заключительный метод полного перебора, задачи которого:

1. Создать всевозможные пары небазисных переменных
2. Перевести с помощью метода смены базиса симплекс таблицу в нужный базис
3. Добавить в массив данных о текущем базисном решение, а именно: Опорное оно или нет, значение целевой функции, текущий базис.
4. Добавлять точку на график

**Приложение**

import numpy as np  
import matplotlib.pyplot as plt  
  
  
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]):  
 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])  
 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 make\_self\_basic(self):  
 for what\_change\_i in range(1, self.rows): # проходим по всему первому столбцу и ищем там y  
 what\_change = self.table[what\_change\_i][0]  
 if 'y' in what\_change:  
 for on\_what\_i in range(1, self.columns): # проходим по всей первой строке и ищем там x  
 on\_what = self.table[0][on\_what\_i]  
 if 'x' in on\_what:  
 break  
 self.change\_basic(row=what\_change\_i, column=on\_what\_i) # меняем x и y  
 self.delete\_column(on\_what\_i) # удаляем колонку с мнимым базисом  
 self.print()  
  
 def delete\_column(self, column\_id):  
 for i in range(self.rows):  
 self.table[i].pop(column\_id)  
 self.columns -= 1  
  
 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 draw\_on\_nonbasic(self):  
 if self.columns != 4:  
 print('Невозможно провести геометрическое решение в таком базисе')  
 return  
 name\_x\_1 = self.table[0][self.columns - 3][2:]  
 name\_x\_2 = self.table[0][self.columns - 2][2:]  
  
 plt.rcParams.update({'font.size': 22})  
 fig = plt.figure()  
 ax = fig.add\_subplot(111)  
 for i in range(1, self.rows - 1):  
 x1 = np.arange(-20, 20, 0.01)  
 b = self.table[i][self.columns - 1]  
 a1 = -self.table[i][self.columns - 3]  
 a2 = -self.table[i][self.columns - 2]  
 x2 = - (b + a1 \* x1) / a2  
 ax.plot(x1, x2, '-',  
 label=f'{round(a1, 3)}\*$x\_{name\_x\_1}$ + ({round(a2, 3)})\*$x\_{name\_x\_2}$ + ({round(b, 3)}) $\geq$ 0')  
  
 if a2 > 0:  
 ax.fill\_between(x1, x2, np.zeros\_like(x2) + 100, alpha=0.5)  
 else:  
 ax.fill\_between(x1, x2, np.zeros\_like(x2) - 100, alpha=0.5)  
 x1 = np.arange(-2, 2, 0.01)  
 b = self.table[-1][self.columns - 1]  
 a1 = -self.table[-1][self.columns - 3]  
 a2 = -self.table[-1][self.columns - 2]  
 x2 = - (b + a1 \* x1) / a2  
 ax.plot(x1, x2, '-',  
 label=f'{round(a1, 3)}\*$x\_{name\_x\_1}$ + ({round(a2, 3)})\*$x\_{name\_x\_2}$ + ({round(b, 3)})=0(c=0)')  
  
 ax.set\_xlabel('x4')  
 ax.set\_ylabel('x5')  
 x = np.arange(-20, 20, 0.05)  
 ax.plot(x, np.zeros\_like(x), '-.')  
 x = np.arange(-100, 100, 0.05)  
 ax.plot(np.zeros\_like(x), x, '-.')  
 ax.legend()  
 self.ax = ax  
 plt.grid()  
  
 def change\_columns(self, on\_what, from\_what):  
 for i in range(self.rows):  
 self.table[i][on\_what], self.table[i][from\_what] = self.table[i][from\_what], self.table[i][on\_what]  
  
 def full\_search(self, draw=True):  
 alphabet = 'A B C D E F G H I J K L M N O P Q R S T U V W X Y Z'.split()  
 pars = []  
 count\_var = self.columns + self.rows - 4  
 for i in range(1, count\_var + 1):  
 for j in range(1, i + 1):  
 if i != j:  
 pars.append((f'x{i}', f'x{j}'))  
  
 self.results = {}  
 for par in pars:  
 for i, var in enumerate(par):  
 i += 1  
 row, column = self.find\_variable(var) # Координаты первого символа в таблица  
 if row == 0:  
 if column != i:  
 self.change\_columns(on\_what=i, from\_what=column)  
 else: # значит переменная не на верхней строке  
 column = i  
 self.change\_basic(row=row, column=column)  
 letter = alphabet[0]  
 alphabet.remove(letter)  
 self.check\_positive()  
 self.results[par] = {'result': self.table[-1][-1], 'Опорное': 'Да' if self.check\_positive() else 'Нет'}  
 print(par, letter)  
 self.print()  
 if draw:  
 self.make\_point(letter)  
  
 print(sorted(sorted(self.results.items(), key=lambda x: x[-1]['result'], reverse=True),  
 key=lambda x: x[-1]['Опорное']))  
 if draw:  
 plt.show()  
  
 def make\_point(self, letter):  
 row, column = self.find\_variable('x4')  
 if row == 0:  
 x4 = 0  
 else:  
 x4 = self.table[row][-1]  
 row, column = self.find\_variable('x5')  
 if row == 0:  
 x5 = 0  
 else:  
 x5 = self.table[row][-1]  
  
 self.ax.annotate(f'{letter}', xy=(x4, x5), xytext=(x4, x5))  
  
 def check\_positive(self):  
 for i in range(1, self.rows - 1):  
 if self.table[i][-1] < 0:  
 return False  
 return True  
  
 def find\_variable(self, name):  
 for i in range(1, self.columns):  
 if name in self.table[0][i]:  
 return 0, i  
 for i in range(1, self.rows):  
 if name in self.table[i][0]:  
 return i, 0  
  
  
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.draw\_on\_nonbasic()  
 table.full\_search()