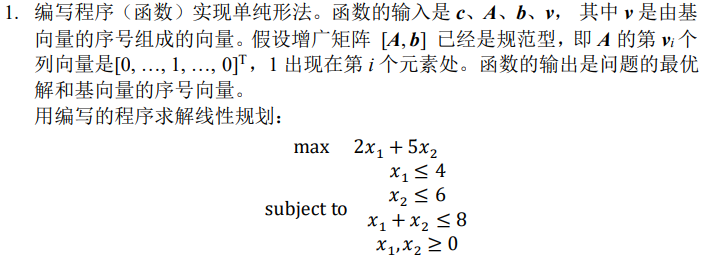
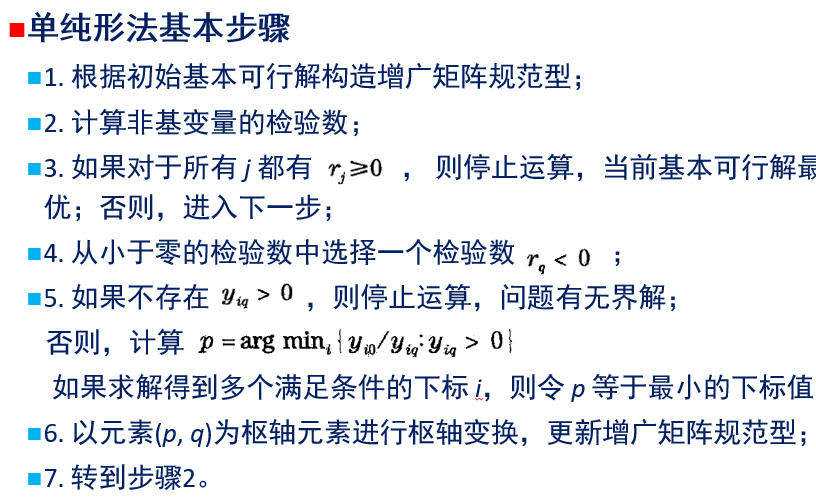
**《最优化理论课程》第三次作业**

**姓名：华浩宇 学号：23020211153893**

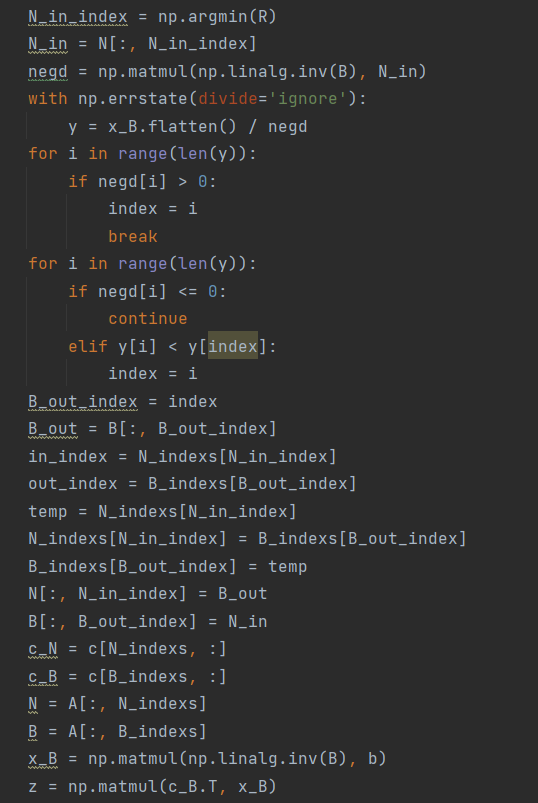


单纯形法的基本步骤：

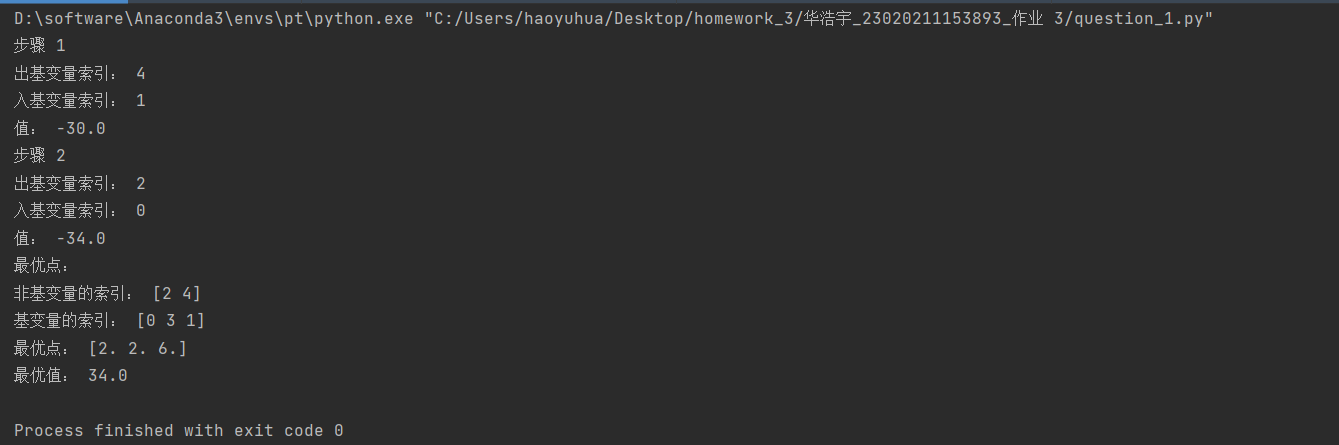


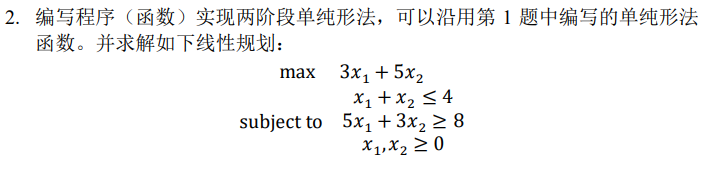
定理：假定某个标准形式的线性规划问题存在最优可行解，利用单纯形法进行求解，当求解过程终止时，最后一步得到的检验数全部非负，那么此时的基本可行解就是最优解。

实现：

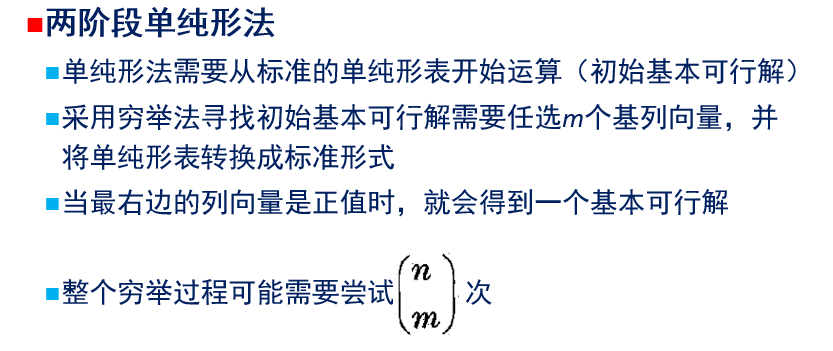


运行结果如下：

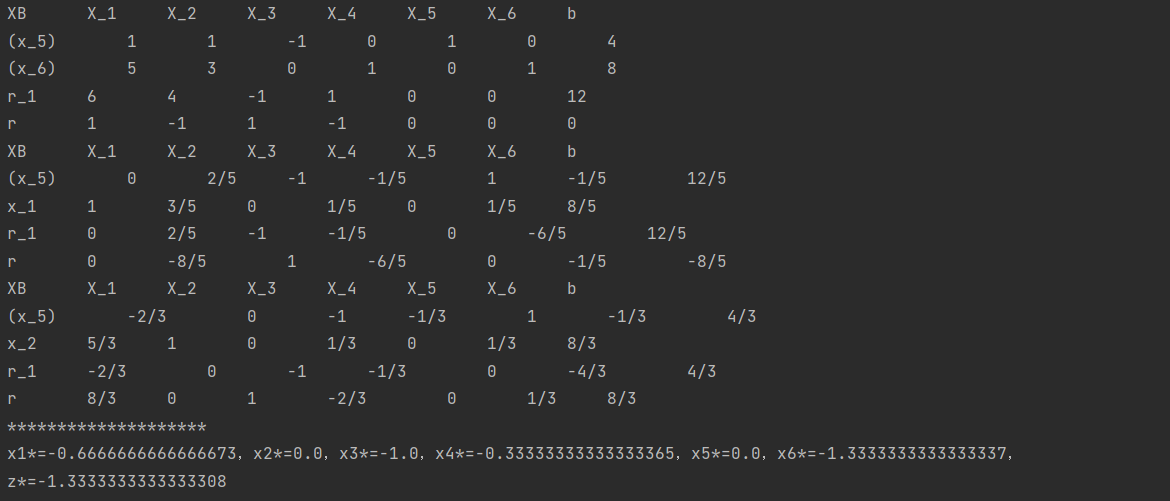




两阶段线性规划和一阶段线性规划的主要区别在于多了一个人为规定基向量方向的过程，用于更好的求解混合约束下的DP问题。



运行结果如下图所示：



代码展示：

问题1:

import numpy as np  
  
  
def simplex(c, A, b):  
 if c.shape[0] != A.shape[1]:  
 print("A和C形状不匹配")  
 return 0  
 if b.shape[0] != A.shape[0]:  
 print("A和b形状不匹配")  
 return 0  
 num = A.shape[1] - A.shape[0]  
 N\_indexs = np.arange(0, num)  
 B\_indexs = np.arange(num, A.shape[1])  
 N = A[:, N\_indexs]  
 B = A[:, B\_indexs]  
 c\_N = c[N\_indexs, :]  
 c\_B = c[B\_indexs, :]  
 x\_B = np.matmul(np.linalg.inv(B), b)  
 j = 0  
 while True:  
 R = (c\_N.T - np.matmul(np.matmul(c\_B.T, np.linalg.inv(B)), N)).flatten()  
 if all(R >= 0):  
 print("最优点：")  
 print("非基变量的索引：", N\_indexs)  
 print("基变量的索引：", B\_indexs)  
 print("最优点：", x\_B.flatten())  
 print("最优值：", -(z.flatten()[0]))  
 return 0  
 else:  
 j = j + 1  
 print("步骤", j)  
 N\_in\_index = np.argmin(R)  
 N\_in = N[:, N\_in\_index]  
 negd = np.matmul(np.linalg.inv(B), N\_in)  
 with np.errstate(divide='ignore'):  
 y = x\_B.flatten() / negd  
 for i in range(len(y)):  
 if negd[i] > 0:  
 index = i  
 break  
 for i in range(len(y)):  
 if negd[i] <= 0:  
 continue  
 elif y[i] < y[index]:  
 index = i  
 B\_out\_index = index  
 B\_out = B[:, B\_out\_index]  
 in\_index = N\_indexs[N\_in\_index]  
 out\_index = B\_indexs[B\_out\_index]  
 temp = N\_indexs[N\_in\_index]  
 N\_indexs[N\_in\_index] = B\_indexs[B\_out\_index]  
 B\_indexs[B\_out\_index] = temp  
 N[:, N\_in\_index] = B\_out  
 B[:, B\_out\_index] = N\_in  
 c\_N = c[N\_indexs, :]  
 c\_B = c[B\_indexs, :]  
 N = A[:, N\_indexs]  
 B = A[:, B\_indexs]  
 x\_B = np.matmul(np.linalg.inv(B), b)  
 z = np.matmul(c\_B.T, x\_B)  
 print("出基变量索引：", out\_index)  
 print("入基变量索引：", in\_index)  
 print("值：", z.flatten()[0])  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 c = np.array([-2, -5, 0, 0, 0]).reshape(-1, 1)  
 A = np.array([[1, 0, 0, 1, 0], [0, 1, 0, 0, 1], [1, 1, 1, 0, 0]])  
 b = np.array([4, 6, 8]).reshape(-1, 1)  
 simplex(c, A, b)

问题2

import numpy as np  
from fractions import Fraction as f  
  
  
def getinput():  
 global m, n  
 m = int(input('输入约束的个数：'))  
 n = int(input('输入变量的个数：'))  
 a = []  
 for i in range(m):  
 a.append(  
 [eval(input('a({}{})='.format(i + 1, j + 1))) for j in range(n)]  
 )  
 e = [list(i) for i in np.diag([1] \* m)]  
 b = [eval(input('b({})='.format(i + 1))) for i in range(m)]  
 r\_1 = []  
 for i in range(n):  
 r\_1.append(sum([a[j][i] for j in range(m)]))  
 r\_1 += [0] \* m  
 r\_1.append(sum(b))  
 r = [eval(input('r({})='.format(i + 1))) for i in range(n)] + [0] \* (m + 1)  
 vect = [n + i + 1 for i in range(m)]  
 a = [i + j + [k] for i, j, k in zip(a, e, b)] + [r\_1] + [r]  
 return a, vect  
  
  
def judge(matrix):  
 if max(matrix[-2][:-1]) <= 0:  
 flag = False  
 else:  
 flag = True  
 return flag  
  
  
def trans():  
 global a, matrix, vect  
 maxi = max(matrix[-2][:-1])  
 index = a[-2][:-1].index(maxi)  
 d = {}  
 for i in a[:-2]:  
 if i[index] > 0:  
 d[i[-1] / i[index]] = a.index(i)  
 pivot = d[min(d)]  
 matrix[pivot] = matrix[pivot] / matrix[pivot][index]  
 for i in range(m + 2):  
 if i != pivot:  
 matrix[i] = matrix[i] - matrix[i][index] \* matrix[pivot]  
 vect[pivot] = index + 1  
 a = [list(i) for i in matrix]  
  
  
def pr(matrix, vect):  
 print('XB', end='\t\t')  
 for i in range(n + m):  
 print('X\_{}'.format(i + 1), end='\t\t')  
 print('b')  
 for i in range(m + 2):  
 if i <= m - 1:  
 if vect[i] > m:  
 print('(x\_{})'.format(vect[i]), end='\t\t')  
 else:  
 print('x\_{}'.format(vect[i]), end='\t\t')  
 elif i == m:  
 print('r\_1', end='\t\t')  
 elif i == m + 1:  
 print('r', end='\t\t')  
 for j in matrix[i]:  
 print(f(str(j)).limit\_denominator(), end='\t\t') # 输出分数形式  
 print(end='\n')  
  
  
def print\_solution(matrix):  
 for i in range(n + m):  
 print('x{}\*={}'.format(i + 1, matrix[-2][i]), end='，')  
 print('\nz\*={}'.format(-matrix[-2][-1]))  
  
  
def main():  
 global a, matrix, vect  
 a, vect = getinput()  
 matrix = np.array(a, dtype=np.float64)  
 pr(matrix, vect)  
 while judge(matrix):  
 trans()  
 pr(matrix, vect)  
 print\_solution(matrix)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()