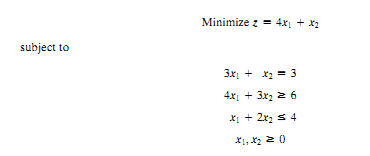
Example 3.4 问题为



可以使用pulp的solve函数，scipy的linprog函数，cvxopt的solvers.lp函数进行求解。

1.对于pulp函数求解该问题可以设置求最大值或者最小值，然后直接把各种约束条件输入进去，函数会自动求出极值点。对于变量的取值范围的约束是通过LpVariable函数的两个bound参数确定的。

源码:

**from** pulp **import** \*  
  
*# 1. 建立问题*prob =pulp.LpProblem(**"Example 3.4.1"**, LpMinimize)  
*# 2. 建立变量*x1 = LpVariable(**"x1"**, 0)  
x2 = LpVariable(**"x2"**, 0)  
  
*# 3. 设置目标函数，第一个式子表示的是求解的目标值*prob += 4\*x1 + x2, **"Z"***# 4. 施加约束条件，可以是等式，也可以是大于等于*

prob += 3\*x1 + x2 == 3  
prob += 4\*x1 + 3\*x2 >= 6  
prob +=x1 + 2\*x2 <= 4  
*# 5. 求解*prob.solve()  
  
*# 6. 打印求解状态*print(**"求解状态:"**, LpStatus[prob.status])  
  
*# 7. 打印出每个变量的最优值***for** v **in** prob.variables():  
 print(v.name, **"="**, v.varValue)  
  
*# 8. 打印最优解的目标函数值*print(**"最优解的目标函数值 = "**, value(prob.objective))

**运行结果：**

求解状态: Optimal

x1 = 0.4

x2 = 1.8

最优解的目标函数值 = 3.4000000000000004

得到最优点取值为x1=0.4,x2=1.8。最小值为3.4

1. 对于使用scipy函数求解此问题时，必须把大于等于号变成小于等于号。同时，scipy函数只支持求解最小值，所以如果求解最大值的时候，就需要在目标函数前面加个负号。但是对于每个变量的取值范围约束比较好写，只需要在bounds里更改就好，比如x1取值为大于0就是（0，None）。

**源码：**

**import** numpy **as** np  
**from** scipy.optimize **import** linprog  
  
c = np.array([4,1])  
A\_ub = np.array([[-4,-3],[1,2]])  
b\_ub = np.array([-6,4])  
A\_eq = np.array([[3,1]])  
b\_eq = np.array([3])  
r = linprog(c,A\_ub,b\_ub,A\_eq,b\_eq,bounds=((0,**None**),(0,**None**)))  
print(r)

**运行结果：**

con: array([-3.77896336e-09])

fun: 3.400000005363207

message: 'Optimization terminated successfully.'

nit: 4

slack: array([1.00000000e+00, 3.63290731e-10])

status: 0

success: True

x: array([0.4, 1.8])

可以看到，所得到的x极值点取值和pulp一致，极值也是取到3.4。

1. 对于cvxopt求解此问题时较为麻烦，首先对于求解极值问题也是只能求最小值，所以对于最大值问题也需要加一个负号。对于不等式约束条件条件，除了要把大于等于乘上负号变成小于等于外，需要把其写成一个G\*X<=h的形式，然后G写成[[-4.,1.,-1.,0],[-3.,2.,0,-1.]]的二维矩阵其中每一行都代表限制条件前对应的变量的系数。同时对于x1>=0这样的约束，应该变成-1\*x1<=0，然后G对应的x1位置为-1，x2对应位置为0。编写约束条件时即为复杂。

源码：

**import** cvxopt  
**from** cvxopt **import** matrix  
c=matrix([4.0,1.0])  
G=matrix([[-4.,1.,-1.,0],[-3.,2.,0,-1.]])  
h=matrix([-6.,4.,0.,0.])  
A=matrix([3.,1.],(1,2))  
b=matrix(3.)  
sv=cvxopt.solvers.lp(c,G,h,A,b)  
print(sv[ 'primal objective'])

print(sv['x'])

运行结果:

3.400000000216262

[ 4.00e-01]

[ 1.80e+00]

结果与其他代码一致。

二：我自己编写的普通单纯形法和两阶段法

**import** numpy **as** np  
*###定义单纯形的类***class** Simplex(object):  
 **def** \_\_init\_\_(self, c, A, b,mode):  
 *# 形式 minf(x)=c.Tx  
 # s.t. Ax=b* self.mode = mode *###True代表求最小值问题，False代表求最大值问题* self.A = A  
 self.b = b  
 A = np.array(A)  
 b = np.array(b)  
 **if** mode==**True**:  
 c=c  
 **else**:  
 c=-1\*c  
 temp = np.hstack((0, c))  
 self.c = c.reshape(-1, 1)  
 temp2 = np.hstack((b, A))  
 self.mat = np.vstack((temp, np.hstack((b, A)))) *###这里的mat就是初等条件下的矩阵* **def** simplex(self):  
 c\_shape = self.c.shape  
 A\_shape = self.A.shape  
 b\_shape = self.b.shape  
 end\_index = A\_shape[1] - A\_shape[0]  
 N = self.A[:, 0:end\_index]  
 Not\_basic = np.arange(0, end\_index) *###Not\_basic是非基变量* c\_Not\_basic = self.c[Not\_basic, :]  
 *# 第一个B必须是可逆的矩阵，其实这里应该用算法寻找，但此处省略* B = self.A[:, end\_index:]  
 B\_columns = np.arange(end\_index, A\_shape[1]) *###B\_columns是基变量* c\_Basic = self.c[B\_columns, :] *###表示去除B\_columns中的指定行* steps = 0  
 **while True**:  
 steps += 1  
 print(**"第 {} 步"**.format(steps))  
 optim\_achieve, B\_columns, Not\_basic = self.main\_simplex(B, N, c\_Basic, c\_Not\_basic, self.b, B\_columns, Not\_basic)  
 **if** optim\_achieve:  
 self.B\_columns = B\_columns  
 self.N\_columns = Not\_basic  
 **break  
 else**:  
 B = self.A[:, B\_columns]  
 N = self.A[:, Not\_basic]  
 c\_Basic = self.c[B\_columns, :]  
 c\_Not\_basic = self.c[Not\_basic, :]  
  
 **def** main\_simplex(self, B, N, c\_B, c\_N, b, B\_columns, N\_columns):  
 B\_inverse = np.linalg.inv(B)  
 P = (c\_N.T - np.matmul(np.matmul(c\_B.T, B\_inverse), N)).flatten()  
 Enter\_variable = np.argmin(P)  
 **if** P.min() >= 0:  
 optim\_achieve = **True** print(**"已经到达极值点"**)  
 best\_solution\_point = np.matmul(B\_inverse, b)  
 print(**"基变量是{}"**.format(B\_columns))  
 print(**"基变量对应的最值点取值是 {}"**.format(best\_solution\_point.flatten()))  
 **if** self.mode==**True**:  
 best\_value=np.matmul(c\_B.T, best\_solution\_point).flatten()[0]  
 **else**:  
 best\_value = -1\*np.matmul(c\_B.T, best\_solution\_point).flatten()[0]  
 print(**"所取到的最值是 {}"**.format(best\_value))  
 **return** optim\_achieve, B\_columns, N\_columns  
 **else**:  
 *# 找到进基变量* Enter\_variable = np.argmin(P)  
 enter = N[:, Enter\_variable].reshape(-1, 1)  
 *# By=Ni， 求出基* y = np.matmul(B\_inverse, enter)  
 pivot\_element = np.matmul(B\_inverse, b)  
 Leave\_variable = self.find\_pivot\_element(y, pivot\_element) *##Leave\_variable代表离基变量的序列号* tmp = N\_columns[Enter\_variable]  
 *##开始进行矩阵运算* mat = self.mat  
 mat = np.array(mat, dtype=**'float64'**)  
 mat[Leave\_variable + 1] = mat[Leave\_variable + 1] / mat[Leave\_variable + 1][  
 Enter\_variable + 1] *##这是因为其实只有两个变量，我补充了4个变量，现在要去掉* ids = np.arange(mat.shape[0]) != Leave\_variable + 1  
 mat[ids] -= mat[Leave\_variable + 1] \* mat[ids,  
 Enter\_variable + 1:Enter\_variable + 2] *# for each i!= row do: mat[i]= mat[i] - mat[row] \* mat[i][col]* print(**"在这一步中对应的单纯形表为"**)  
 print(mat)  
 self.mat = mat  
 N\_columns[Enter\_variable] = B\_columns[Leave\_variable]  
 B\_columns[Leave\_variable] = tmp  
 optim\_achieve = **False** print(**"没有到达极值点"**)  
 print(**"基变量是{0}，非基变量是{1}"**.format(sorted(B\_columns), sorted(N\_columns)))  
 **return** optim\_achieve, B\_columns, N\_columns  
  
 *###找到枢轴元素所在的位置，通过比较x\_B/y最小且y>0的位置* **def** find\_pivot\_element(self, y, pivot\_element):  
 index = []  
 min\_value = []  
 **for** i, value **in** enumerate(y):  
 **if** value <= 0:  
 **continue  
 else**:  
 index.append(i)  
 min\_value.append(pivot\_element[i] / float(value))  
 temp=np.argmin(min\_value)  
 actual\_index = index[temp]  
 **return** actual\_index  
*###定义普通单纯形法函数***def** Common\_simplex(A,b,c,mode):  
 simplex = Simplex(c, A, b,mode)  
 simplex.simplex()  
*###定义二阶段法函数***def** Two\_steps(variable\_true,aritifical\_variable,A,b,c,obj,mode):  
 *##这里的变量个数表示不含人工变量的个数* simplex = Simplex(c, A, b,**True**)*##这里的mode是固定的，因为第一步都是求最小值问题* simplex.simplex()  
 B\_column = simplex.B\_columns  
 N\_column = simplex.N\_columns  
 B\_column = B\_column[B\_column < variable\_true]  
 N\_column = N\_column[N\_column < variable\_true]  
 *###一直到这里一阶段解决了，开始进行二阶段  
 ###目标是去掉R1列和R2列和第一行* obj\_row=np.arange(variable\_true+1,variable\_true+aritifical\_variable+1)*###这里的obj\_row就是R1,和R2所在列，人工变量列都放在了最后* mat\_temp = np.delete(simplex.mat, obj\_row, axis=1)  
 mat\_temp = np.delete(mat\_temp, [0], axis=0)  
 *##这样得到的mat\_temp是去掉人工变量的，下面加上新的目标函数  
 ###print(Two\_steps.b)  
 ###print(Two\_steps.mat[0])* b\_temp = mat\_temp[:, 0].reshape(-1, 1)  
 A\_temp = np.delete(mat\_temp, 0, axis=1)  
 Two\_steps = Simplex(obj, A\_temp, b\_temp,mode)  
 B = Two\_steps.A[:, B\_column]  
 N = Two\_steps.A[:, N\_column]  
 c\_B = Two\_steps.c[B\_column, :]  
 c\_N = Two\_steps.c[N\_column, :]  
 steps = 0  
 **while True**:  
 steps += 1  
 print(**"第 {} 步"**.format(steps))  
 is\_optim, B\_columns, N\_columns = Two\_steps.main\_simplex(B, N, c\_B, c\_N, Two\_steps.b, B\_column, N\_column)  
 **if** is\_optim:  
 **break  
 else**:  
 B = Two\_steps.A[:, B\_columns]  
 N = Two\_steps.A[:, N\_columns]  
 c\_B = Two\_steps.c[B\_columns, :]  
 c\_N = Two\_steps.c[N\_columns, :]  
*###定义主函数***if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *###这里解答Example 3.3* mode\_common=**False***###题目为解决最大值问题，因此mode为False* c\_common = np.array([5, 4, 0, 0, 0,0])  
 A\_common = np.array([[6, 4, 1, 0,0, 0], [1, 2, 0, 1,0,0], [-1, 1, 0, 0, 1,0],[0, 1, 0, 0, 0,1]])  
 b\_common = np.array([24, 6, 1,2]).reshape(-1, 1)  
 Common\_simplex(A\_common,b\_common,c\_common,mode\_common)  
 print(**"Example 3.3 解答完毕"**)  
 *###这里解答Example 3.4* c = np.array([0, 0, 0, 0, 1, 1])  
 A = np.array([[3, 1, 0, 0, 1, 0], [4, 3, -1, 0, 0, 1], [1, 2, 0, 1, 0, 0]])  
 b = np.array([3, 6, 4]).reshape(-1, 1)  
 obj = np.array([4, 1, 0, 0])  
 variable\_true=4*###这个表示原有变量个数* aritifical\_variable=2*##这个表示人工变量的个数* Two\_mode=**True** Two\_steps(variable\_true,aritifical\_variable,A,b,c,obj,Two\_mode)  
 print(**"Example 3.4 解答完毕"**)

我是定义类的方法定义了一个可以求解最大值或者最小值的普通单纯形的类，mode=True代表求解最大值。在主函数中直接传参给Common\_simplex就可以直接求解普通单纯形问题。对于两阶段法，我是在普通单纯形的基础上修改的，因为一阶段求解的是最小值问题，所以mode默认取False，一阶段得到的单纯形表经过删除人工变量和替换目标函数后，再去求解二阶段。

运行结果：

第 1 步

在这一步中对应的单纯形表为

[[20. 0. -0.66666667 0.83333333 0. 0.

0. ]

[ 4. 1. 0.66666667 0.16666667 0. 0.

0. ]

[ 2. 0. 1.33333333 -0.16666667 1. 0.

0. ]

[ 5. 0. 1.66666667 0.16666667 0. 1.

0. ]

[ 2. 0. 1. 0. 0. 0.

1. ]]

没有到达极值点

基变量是[0, 3, 4, 5]，非基变量是[1, 2]

第 2 步

在这一步中对应的单纯形表为

[[21. 0. 0. 0.75 0.5 0. 0. ]

[ 3. 1. 0. 0.25 -0.5 0. 0. ]

[ 1.5 0. 1. -0.125 0.75 0. 0. ]

[ 2.5 0. 0. 0.375 -1.25 1. 0. ]

[ 0.5 0. 0. 0.125 -0.75 0. 1. ]]

没有到达极值点

基变量是[0, 1, 4, 5]，非基变量是[2, 3]

第 3 步

已经到达极值点

基变量是[0 1 4 5]

基变量对应的最值点取值是 [3. 1.5 2.5 0.5]

所取到的最值是 21.0

Example 3.3 解答完毕

第 1 步

在这一步中对应的单纯形表为

[[ 0. 0. 0. 0. 0. 1. 1. ]

[-1.5 0. -1.25 0.75 0. 1. -0.75]

[ 1.5 1. 0.75 -0.25 0. 0. 0.25]

[ 2.5 0. 1.25 0.25 1. 0. -0.25]]

没有到达极值点

基变量是[0, 3, 5]，非基变量是[1, 2, 4]

第 2 步

在这一步中对应的单纯形表为

[[ 0. 0. 0. 0. 0. 1. 1. ]

[ 1. 0. 0. 1. 1. 1. -1. ]

[ 0. 1. 0. -0.4 -0.6 0. 0.4]

[ 2. 0. 1. 0.2 0.8 0. -0.2]]

没有到达极值点

基变量是[0, 1, 3]，非基变量是[2, 4, 5]

第 3 步

已经到达极值点

基变量是[3 0 1]

基变量对应的最值点取值是 [1. 0.6 1.2]

所取到的最值是 0.0

第 1 步

在这一步中对应的单纯形表为

[[-inf nan nan -inf -inf]

[ inf nan nan inf inf]

[-inf nan nan -inf -inf]

[ nan nan nan nan nan]]

没有到达极值点

基变量是[0, 1, 2]，非基变量是[3]

第 2 步

已经到达极值点

基变量是[2 0 1]

基变量对应的最值点取值是 [1. 0.4 1.8]

所取到的最值是 3.4000000000000004

Example 3.4 解答完毕

可以看到对于3.4的解答求得的极值点为x1=0.4,x2=1.8，最值为3.4，与其他函数求解结果一致。