Project 1 - Standard Form of Linear Programming

1930026143　　薛劭杰

1. (a) We can find that there is not any inequation, transform the problem to the standard form of linear programming, we should maximize the $29x_1 + 45x_2$. Because we want to maximize but not minimize, what we need to do is take the negative of the original, and then take the negative of the final result. So we have $c = [-29, -45, 0, 0]$.

Then which we should pay attention to that $A_{ub} \leq b_{ub}$, which means that if there are some " $\geq$ ", we should multiply both sides by -1 to transform the " $\geq$ " to " $\leq$ ".

$A_{ub} =$ `[[ 1, -1, -3,  0],`
`[-2,  3,  7, -3]]`

$b_{ub} = [5, -10]^T$

$A_{eq} =$ `[[2, 8, 1, 0],`
`[4, 4, 0, 1]]`

$b_{eq} = [60, 60]^T$

$l = [0, 0, -\infty, -3]^T$

$u = [\infty, 5, 0.5, \infty]^T$

(b) The code and the compilation result as follows:

In [30]:
```
c = np.array([-29, -45, 0, 0])
A_ub = np.array([[1, -1, -3, 0], [-2, 3, 7, -3]])
b_ub = np.array([5, -10])
A_eq = np.array([[2, 8, 1, 0], [4, 4, 0, 1]])
b_eq = np.array([60, 60])
bound_1 = (0, None)
bound_2 = (0, 5)
bound_3 = (None, 0.5)
bound_4 = (-3, None)
bounds = (bound_1, bound_2, bound_3, bound_4)
linprog(c,A_ub,b_ub,A_eq,b_eq,bounds=bounds)
```

Out[30]:
```
     con: array([15.5361182 , 16.61287594])
     fun: -370.232236243415
 message: 'The algorithm terminated successfully and determined that the problem is infeasible.'
     nit: 6
   slack: array([ 0.79315048, -1.76308648])
  status: 2
 success: False
       x: array([ 6.60059421,  3.97366676, -0.52664069,  1.09008018])
```

In [32]:
```
-linprog(c,A_ub,b_ub,A_eq,b_eq,bounds=bounds).fun
```

Out[32]:
```
370.232236243415
```

In the end, remember to take the negative of the final result which is the real maximum value that solve. However, in this problem, it has no solution.

2. (a) We can find that there is not any inequation, transform the problem to the standard form of linear programming, we should maximize the $x_1 + 4x_2 + x_3$. Because we want to maximize but not minimize, what we need to do is take the negative of the original, and then take the negative of the final result. So we have $c = [-1, -4, -1]$.

$A_{ub} = None$ and $b_{ub} = None$

$A_{eq} =$ `[[ 2, -2,  3],`
`[ 1,  0, -1]]`

$b_{eq} = [4, 1]^T$
$l = [-\infty,\ 0,\ 0]^T$
$u = [\infty, \infty, \infty]^T$
(b) The code and the compilation result as follows:

```
In [23]: c = np.array([-1, -4, -1])
         # A_ub = np.array([])
         # b_ub = np.array([])
         A_eq = np.array([[2, -2, 3], [1, 0, -1]])
         b_eq = np.array([4, 1])
         bounds = ((None, None), (0, None), (0, None))
         linprog(c,A_ub=None,b_ub=None,A_eq=A_eq,b_eq=b_eq,bounds=bounds)

Out[23]:      con: array([4.72939301, 3.15248871])
              fun: -611023516879.9122
          message: 'The algorithm terminated successfully and determined that the problem is unbounded.'
              nit: 4
            slack: array([], dtype=float64)
           status: 3
          success: False
                x: array([5.09186264e+10, 1.27296566e+11, 5.09186264e+10])

In [24]: -linprog(c,A_ub=None,b_ub=None,A_eq=A_eq,b_eq=b_eq,bounds=bounds).fun

Out[24]: 611023516879.9122
```

In the end, remember to take the negative of the final result which is the real maximum value that solve. However, in this problem, it has no solution.

3. (a) We can find that there is not any equation, so transform the problem to the standard form of linear programming, we should minimize the $x + 2y + 3z$ and we have: $c = [1, 2, 3]$. Then which we should pay attention to that $A_{ub} \leq b_{ub}$, which means that if there are some " $\geq$ ", we should multiply both sides by -1 to transform the " $\geq$ " to " $\leq$ ".

$$A_{ub} = \begin{matrix}[[\ 1,\ \ 1,\ \ 0],\\ [-1,\ -1,\ \ 0],\\ [\ 1,\ \ 0,\ \ 1],\\ [-1,\ \ 0,\ -1]]\end{matrix}$$

$b_{ub} = [3, -2, 5, -4]^T$
$A_{eq} = None\ \ and\ \ b_{eq} = None$
$l = [0, 0, 0]^T$
$u = [\infty, \infty, \infty]^T$

(b) The code and the compilation result as follows:

In [27]:
```
c = np.array([1, 2, 3])
# A_ub = np.array([])
# b_ub = np.array([])
A_ub = np.array([[1, 1, 0], [-1, -1, 0], [1, 0, 1], [-1, 0, -1]])
b_ub = np.array([3, -2, 5, -4])
bounds = ((0, None), (0, None,), (0, None))
linprog(c,A_ub=A_ub,b_ub=b_ub,A_eq=None,b_eq=None,bounds=bounds)
```

Out[27]:
```
     con: array([], dtype=float64)
     fun: 5.999999993369687
 message: 'Optimization terminated successfully.'
     nit: 4
   slack: array([ 5.72031533e-09,  9.99999994e-01,  1.00000001e+00, -8.55915161e-09])
  status: 0
 success: True
       x: array([2.99999999e+00, 1.90162796e-09, 9.99999999e-01])
```

We can see that the minimum result of $x + 2y + 3z$ is 5.999999993369687 when $x =$ [2.99999999e+00, 1.90162796e-09, 9.99999999e-01]

4. (a) We know that there two different plant, two different source and three different market. So we can have 12 kinds of choice. Set that $A_{11}, A_{12}, A_{13}, A_{21}, A_{22}, A_{23}, B_{11}, B_{12}, B_{13}, B_{21}, B_{22}, B_{23}$. The character A or B indicates which plant, the first subscript indicates which source and the second subscript indicates which market transport to. Our goal is to select the best combination to minimize transportation cost. We have some limiting conditions which can be expressed by the inequation:

1. $A_{11} + A_{12} + A_{13} + B_{11} + B_{12} + B_{13} \leq 10$ (10 tons are available from source 1)
2. $A_{21} + A_{22} + A_{23} + B_{21} + B_{22} + B_{23} \leq 15$ (15 tons from source 2)
3. $A_{11} + A_{21} + B_{11} + B_{21} \geq 8$ (Market centers 1 require 8 tons)
4. $A_{12} + A_{22} + B_{12} + B_{22} \geq 14$ (Market centers 2 require 14 tons)
5. . $A_{13} + A_{23} + B_{13} + B_{23} \geq 3$ (Market centers 3 require 3 tons)

Then we should minimize the transportation cost. According to the two table, we can calculate the total transportation costs for each option. Therefore, we can get:

$$\min \quad (500 * A_{11} + 300 * A_{12} + 200 * A_{13} + 600 * A_{21} + 400 * A_{22} + 300 * A_{23} + 450 * B_{11} + 550 * B_{12} + 350 * B_{13} + 450 * B_{21} + 550 * B_{22} + 350 * B_{23})$$

Then transform the problem to the standard form of linear programming:

$c =$ [500, 300, 200, 600, 400, 300, 450, 550, 350, 450, 550, 350]^T

$A_{ub} =$
```
[[ 1,  1,  1,  0,  0,  0,  1,  1,  1,  0,  0,  0],
 [ 0,  0,  0,  1,  1,  1,  0,  0,  0,  1,  1,  1],
 [-1,  0,  0, -1,  0,  0, -1,  0,  0, -1,  0,  0],
 [ 0, -1,  0,  0, -1,  0,  0, -1,  0,  0, -1,  0],
 [ 0,  0, -1,  0,  0, -1,  0,  0, -1,  0,  0, -1]]
```

$b_{ub} =$ [10, 15, -8, -14, -3]^T

$A_{eq} = None \ and \ b_{eq} = None$

$l = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$

$u = [\infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty,]^T$

(b) The code and the compilation result as follows:

In [4]:
```
c = np.array([500,300,200,600,400,300,
              450,550,350,450,550,350])
A_ub = np.array([[1,1,1,0,0,0,1,1,1,0,0,0], [0,0,0,1,1,1,0,0,0,1,1,1],
                 [-1,0,0,-1,0,0,-1,0,0,-1,0,0], [0,-1,0,0,-1,0,0,-1,0,0,-1,0],
                 [0,0,-1,0,0,-1,0,0,-1,0,0,-1]])
b_ub = np.array([10,15,-8,-14,-3])
bounds = ((0, None), (0, None), (0, None), (0, None), (0, None), (0, None),
          (0, None), (0, None), (0, None), (0, None), (0, None), (0, None))
linprog(c,A_ub=A_ub,b_ub=b_ub,A_eq=None,b_eq=None,bounds=bounds)
```

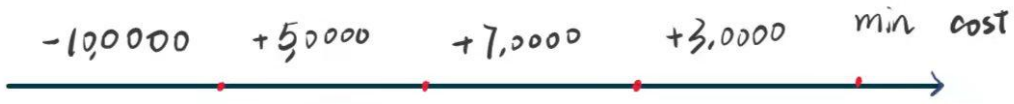
Out[4]:
```
     con: array([], dtype=float64)
     fun: 9099.999999799946
 message: 'Optimization terminated successfully.'
     nit: 6
   slack: array([ 1.65373493e-10,  4.47183623e-10, -1.93640659e-10, -4.32809344e-10,
        1.38919987e-11])
  status: 0
 success: True
       x: array([3.00581178e-11, 8.44138162e+00, 1.55861838e+00, 3.23375399e-11,
       5.55861838e+00, 1.44138162e+00, 4.33977514e-11, 2.31537793e-11,
       3.92658490e-11, 8.00000000e+00, 3.48923046e-11, 1.77807012e-10])
```

We can see that the minimum transportation cost is 9099.999999799946

when $x =$ [3.00581178e-11, 8.44138162e+00, 1.55861838e+00, 3.23375399e-11, 5.55861838e+00, 1.44138162e+00, 4.33977514e-11, 2.31537793e-11, 3.92658490e-11, 8.00000000e+00, 3.48923046e-11, 1.77807012e-10]

5.

Assume that we can loan and borrow in each period. And we should make the Interest minimization to obtain the most fund eventually. We have a time line as follow:



Then we should make sure that all the operation should be finished in the fourth period. (The loan and interest are collected and the loan must be paid off). In this question, we should also satisfy the condition that the fund cannot be negative anytime. So we can get some relation function, X is borrow and L is loan. The first number of subscripts is which number and the second one is which kind of borrow (n-$period$)

1. For the first period, we can borrow for two-period (22%) and three-period (34%) but not one period (12%) because it is not some earning after one period, so at that time the borrow must not be paid off. And that time we cannot loan to other (10%), because it will loss money by low interest.

$$X_{12} + X_{13} \geq 100000\#(1)$$

2. We can borrow for one-period (12%), two-period (22%) and three-period (34%). And also can load to others, we should add the rest fund of the first period.

$$(X_{12} + X_{13} - 100000) + 50000 + X_{21} + X_{22} + X_{23} - L_2 \geq 0\ \#(2)$$

3. We can borrow for one-period (12%) and two-period (22%) but not three-period because of the limitation of the time. And also can load to others.

$$(2) + 70000 + X_{31} + X_{32} - 1.12X_{21} - 1.22X_{12} + 1.1L_2 - L_3 \geq 0\#(3)$$

4. We can borrow for one-period (12%) but not two-period and three-period because of the limitation of the time. And also cannot load to others.

$$(3) + 30000 - 1.12X_{31} - 1.34X_{13} - 1.22X_{22} + X_{41} + 1.1L_3 - L_4 \geq 0 \ \#(4)$$

So we can simplify this transform the problem to the standard form of linear programming:

$c = [0.22,\ 0.34, 0.12,\ 0.22,\ 0.34,\ 0.12,\ 0.22,\ 0.12,\ 0,\ 0,\ 0]^T$

$A\_ub =$
```
[[-1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0.22, -1, 0.12, -1, -1, 0, 0, 0, -0.1, 0, 0],
 [0.22, 0.34, 0.12, 0.22, -1, 0.12, -1, 0, -0.1, -0.1, 0],
 [0.22, 0.34, 0.12, 0.22, 0.34, 0.12, 0.22, 0.12, -0.1, -0.1, -0.1]]
```

$b_{ub} = [-100000,\ -50000,\ 20000,\ 50000]^T$

$A_{eq} = None$ and $b_{eq} = None$

$l = [0,0,0,0,0,0,0,0,0,0,0]^T$

$u = [\infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty]^T$

And the code is as follow:

In [13]:
```
c  = [0.22, 0.34,0.12, 0.22, 0.34, 0.12,0.22,0.12,0,0,0]
A_ub = [[-1,-1,0,0,0,0,0,0,0,0,0],
        [0.22,-1,0.12,-1,-1,0,0,0,-0.1,0,0],
        [0.22,0.34,0.12,0.22,-1,0.12,-1,0,-0.1,-0.1,0],
        [0.22,0.34,0.12,0.22,0.34,0.12,0.22,0.12,-0.1,-0.1,-0.1]]
b_ub = [-100000,-50000,20000,50000]
A_eq = None
b_eq = None
bound=[(0,None)]*11
linprog(c,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq, b_eq=b_eq, bounds=bound)
```

Out[13]:
```
     con: array([], dtype=float64)
     fun: 22000.00000141477
 message: 'Optimization terminated successfully.'
     nit: 12
   slack: array([3.10392352e-07, 7.99067467e+03, 8.26865980e+04, 1.21720155e+05])
  status: 0
 success: True
       x: array([1.00000000e+05, 7.10740326e-06, 1.62446569e-06, 3.22564030e-08,
       7.29545189e-08, 8.44218954e-07, 2.87582088e-07, 8.51520195e-07,
       7.99906747e+05, 4.69592333e+04, 9.03355741e+04])
```

In this way we get just cost 22000.00000141477 eventually which mean that we can get about 28000 as maximum when

```
x: array([1.00000000e+05, 7.10740326e-06, 1.62446569e-06, 3.22564030e-08,
7.29545189e-08, 8.44218954e-07, 2.87582088e-07, 8.51520195e-07,
7.99906747e+05, 4.69592333e+04, 9.03355741e+04])
```