Optimization basics: Linear Programming

Author: Igor dos Santos Montagner
Date: 03/02/2015

One of the Optimization topics I have been reviewing is *Linear Programming*. In this category of optimization problems, both the cost function and all the restrictions are linear.

Although there are many ways to write a linear programming problem, the formulation below, called *Standard Form*, is used by many books and tools in Optimization.

$$\begin{cases} \text{minimize} f(x) = c^T x \\ \text{ s.a.} \\ Ax = b \\ x \geq 0 \end{cases}$$

Converting a problem to *Standard Form*

Any LP problem can be converted into the *Standard Form* using the transformations below:

1. If the problem is a maximization one, convert it to minimize $-f(x)$;
2. If there are constraints of $\geq$, add a variable s_i for each such constraint such that $A_i x - s_i = b$ and $s_i \geq 0$;
3. If there are constraints of $\leq$, add a variable s_i for each such constraint such that $A_i x + s_i = b$ and $s_i \geq 0$;
4. If there are unbounded variables, replace them by two non-negative variables $x_i = x_i^{(+)} - x_i^{(-)}, x_i^{(+)}, x_i^{(-)} \geq 0$.

Transformations (1) and (4) are relatively simple, but transformations (2) and (4) are a little more involved. In these cases s_i is not part of the cost function, so if during the optimization the restriction $A_i x = b$ is active, assigning $s_i = 0$ does not impact neither the cost function neither the actual feasible set.

The Dual problem

Every minimization problem (*primal*) in LP has a related maximization *dual* problem such that both have the same optimal cost function value and it is possible to retrieve the solution from one problem from the other (and vice-versa).

Given a LP in the standard form, its dual is given by

$$\begin{cases} \text{maximize} f(y) = b^T y \text{ s.a.} \\ A^T y \leq c \end{cases}$$

Let $u = c - A^T y$. This vector contains the "distance" from each constraint to the equality. A constraint only effectively constrains the solution if it is saturated, in other words, if $u_i = 0$. Therefore, we can ignore every column of A such that $u_j \neq 0$, since the dual constraint is not active. Se the dual problem has a unique solution, the number of non-zero components of u will be equal to the number of rows of A and we can calculate x by solving the following linear system:

$$A[:, u_i = 0] x^* = b$$

.

The positions of x such that $u_i \neq 0$ are completed with $x^* = 0$, since all other constraints are already satisfied by the other components of x_i^*.

Practical example

Enunciado:

> A startup want to build talking washing machines spending the least possible. There are three ways of building them: manually, semi-automatically and automatically. The manual production demands 1 minute of qualified work, 40 minutes of non-qualified work and three minutes of assemblage. The work times are 4, 30 and 2 minutos for the semi-automatic method and 8, 20 and 4 minutos for the fully automatic method. A startup has a pool of 4500 minutes of qualified work, 36000 minutos of non-qualified work and 2700 minutos of assembly. The costs of the production are 70, 80 and 85 euros for the manual, semi-automatic and automatic methods.

1. Write the problem above using LP and give a solution

The variables are the number of machines $x_1, x_2 e x_3$ built using each method(manual, semi-automatically and automatically). The cost to be minimized is the production cost: $70x_1 + 80x_2 + 85x_3$. There are constraints regarding the number of machines to be produced (999) and the capacity of the factory. The complete formulation is show below.

$$\begin{cases} \text{minimize} f(x) = 70x_1 + 80x_2 + 85x_3 \text{ s.a.} \\ x_1 + x_2 + x_3 = 999 \\ x_1 + 4x_2 + 8x_3 \leq 4500 \\ 40x_1 + 30x_2 + 20x_3 \leq 36000 \\ 3x_1 + 2x_2 + 4x_3 \leq 2700 \\ x \geq 0 \end{cases}$$

Solution using scipy:

```
import numpy as np
from scipy.optimize import linprog
from numpy.linalg import solve

A_eq = np.array([[1,1,1]])
b_eq = np.array([999])

A_ub = np.array([
[1, 4, 8],
[40,30,20],
[3,2,4]])

b_ub = np.array([4500, 36000,2700])

c = np.array([70, 80, 85])

res = linprog(c, A_eq=A_eq, b_eq=b_eq, A_ub=A_ub, b_ub=b_ub,
bounds=(0, None))
print('Optimal value:', res.fun, '\nX:', res.x)
```

```
Optimal value: 73725.0
X: [ 636.  330.   33.]
```

2. Convert the problem into the standard form

It is necessary to add slack variables for all inequality constraints. The problem becomes;

$$\begin{cases} \text{minimize} f(x) = 70x_1 + 80x_2 + 85x_3 \text{ s.a.} \\ x_1 + x_2 + x_3 = 999 \\ x_1 + 4x_2 + 8x_3 + s_1 = 4500 \\ 40x_1 + 30x_2 + 20x_3 + s_2 = 36000 \\ 3x_1 + 2x_2 + 4x_3 + s_3 = 2700 \\ x \geq 0, s \geq 0 \end{cases}$$

Solution using scipy:

```
A = np.array([
[1, 1, 1, 0, 0, 0],
[1, 4, 8, 1, 0, 0],
[40, 30, 20, 0, 1, 0],
[3, 2, 4, 0, 0, 1]])

b = np.array([999, 4500, 36000, 2700])
c = np.array([70, 80, 85, 0, 0, 0])

res = linprog(c, A_eq=A, b_eq=b, bounds=(0, None))
print('Optimal value:', res.fun, '\nX:', res.x)
```

```
Optimal value: 73725.0
X: [  636.   330.    33.  2280.     0.     0.]
```

Note that s_1 = 2280$implies that$x_1 + 4x_2 + 8x_3 \leq 4500$.

3. Present the dual of the problem above and show that both problems achieve the same results

The dual problem is

$$\begin{cases} \text{maximize} f(y) = 999y_1 + 4500y_2 + 360000y_3 + 2700y_4 \text{ s.a.} \\ y_1 + y_2 + 40y_3 + 3y_4 \leq 70 \\ y_1 + 4y_2 + 30y_3 + 2y_4 \leq 80 \\ y_1 + 8y_2 + 20y_3 + 4y_4 \leq 85 \\ y_2 \leq 0 \\ y_3 \leq 0 \\ y_4 \leq 0 \end{cases}$$

Solution using scipy: We use the variables declared previously in this solution. **Note that since *linprog* only solves minimization problems, that sign of the cost function is inverted.**

```
res = linprog(-b, A_ub=A.T, b_ub=c, bounds=[(None,None), (None,None),
(None,None), (None,None)])
y = res.x
print('Optimal value:', -res.fun, '\nY:', y)

u = c - A.T.dot(y)
Ar = A[:, np.abs(u)< 1e-10]
x_1 = solve(Ar, b)
print(x_1)
x = np.array([0.0] * len(c))
x[np.abs(u)< 1e-10] = x_1
x[np.abs(u)> 1e-10] = 0
print('Primal solution from the dual:', x)
```

```
Optimal value: 73725.0
Y: [ 108.33333333    0.            -0.83333333   -1.66666667]
[  636.   330.    33.  2280.]
Primal solution from the dual: [  636.   330.    33.  2280.     0.
0.]
```

Made with Pweave and Pandoc.