MATH404: Report 1

Revised Simplex Method for solving Linear Programming Problems

By:

Abdelrahman Alaa Elaraby | 201700556

1. Motivation

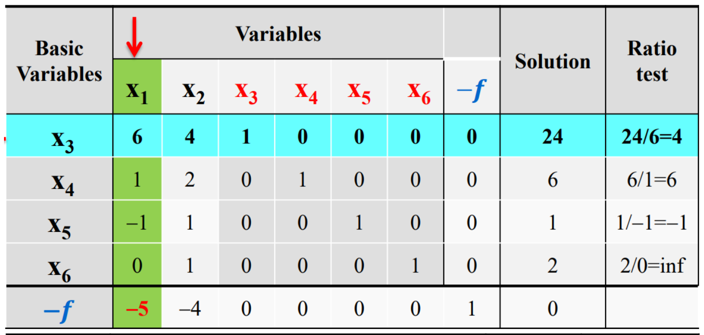
A class of optimization/programming problems is known as “Linear programming”. In that class both objective function and constraints are linear functions in the design/decision variables. The given constraints might be in the form of equalities or inequalities. Though it might seem simple, yet it solves many problems in the real world industry. The linear programming class of problems was first recognized by economists in the 1930s while trying to find out the optimal allocation of resources in some case studies. During World War II, George B. Dantzig, a member of the US Air Forces back then, designed the general framework of the linear programming problem and created the simplex algorithm in 1947.

Despite the fact that the simplex Algorithm is capable of solving linear programming problems better than the brute force algorithm that searches the space of all extreme points (vertices), yet it requires a lot of memory and computational time when the dimension of the problem grows (number of variables and constraints). And for that purpose - efficiency on bigger problems - later adjustments on the simplex algorithm were made, and one of them is the revised simplex method which will be discussed in that report.

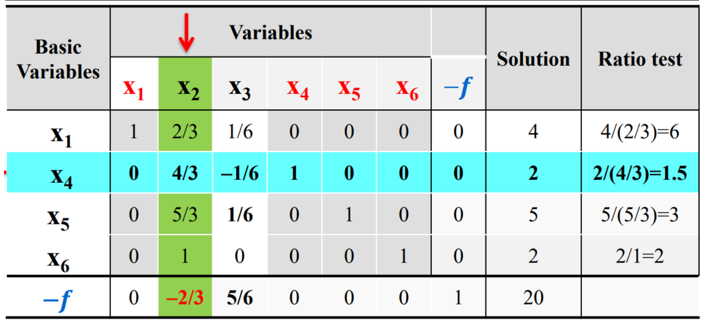
2. Theory & Algorithm

The basic simplex method retains much unused information in its simplex tableau. And operates on those unused information doing unnecessary processing. Making it less time and memory efficient.

For illustration, in the table on the right, columns (x4,x5,x6) remained unchanged. In a larger tableau, the **sparsity** will be more pronounced, many zeros, many unchanged values, thus more unnecessary storage and computations.



Basic Variables	Variables							Solution	Ratio test
	x_1	x_2	x_3	x_4	x_5	x_6	$-f$		
x_3	**6**	**4**	**1**	**0**	**0**	**0**	**0**	**24**	**24/6=4**
x_4	1	2	0	1	0	0	0	6	6/1=6
x_5	–1	1	0	0	1	0	0	1	1/–1=–1
x_6	0	1	0	0	0	1	0	2	2/0=inf
$-f$	–5	–4	0	0	0	0	1	0	



Basic Variables	Variables							Solution	Ratio test
	x_1	x_2	x_3	x_4	x_5	x_6	$-f$		
x_1	1	2/3	1/6	0	0	0	0	4	4/(2/3)=6
x_4	**0**	**4/3**	**–1/6**	**1**	**0**	**0**	**0**	**2**	**2/(4/3)=1.5**
x_5	0	5/3	**1/6**	0	1	0	0	5	5/(5/3)=3
x_6	0	1	0	0	0	1	0	2	2/1=2
$-f$	0	–2/3	**5/6**	0	0	0	1	20	

If the steps of the simplex algorithm were followed closely, it will be noticed that the necessary information needed to change the tableau are the following:
- The non-basic variables' columns.
- The non-basic variables' coefficients in the objective function (candidate entering variables)
- The right hand side (RHS), or the values of the basic variables.
If that kept in mind, the problem can be formulated in the following way, using *decomposition principle*:

$\mathbf{BX_{basic} + NX_{nonbasic} = b}$

Where **B** is the matrix multiplied by the basic variables, **N** is matrix multiplied by the non-basic variables, **b** is the RHS of the system of equations. The non-basic variables are chosen to be zero, thus the system of equations can be written as:

$\mathbf{BX_{basic} = b}$
Thus: $\mathbf{X_{basic} = B^{-1}b}$

The values of the next basic variables can be known by multiplying the inverse of the base matrix by the existing right hand side.
Steps can be summarized as follows:
1-The base matrix is constructed from the original matrix and by knowing the basic variables
2-calculate the new column vectors of the non-basic variables by multiplying B^{-1} by current columns, named $\mathbf{P}_i$.

$$\mathbf{P}_{i} = \mathbf{B^{-1}P}_i$$

3-calculate the new values of the basic variables by multiplying B^{-1} by current RHS.
4-calculate the new non-basic variables' coefficients of the objective function by:

$$\mathbf{C_i = C_i - C_{basis}P_i}$$

Where, $\mathbf{C_{basis}}$ is the vector containing the current coefficients of basic variables, and $\mathbf{P}_i$ is the current values of the non-basic columns.
5-Pick the variable "i" that corresponds to the most negative C_i in the objective function coefficients. This variable will be the new basic variable, then repeat from step one till no -ve $\mathbf{C_i}$.

4. Implementation

The discussed algorithm was implemented using python, and numpy library, from scratch. I went through the coding by dividing the problem into 3 functions:

# function 1: *put_to_std_form(c, A_ub=[], b_ub=[], A_eq=[], b_eq=[])*:

This function puts the input in the standard canonical form, by addition of slack/surplus/artificial variables when needed. It also determines the indices of A.Vs and their Sum "W".

INPUTS:

c: coefficients of Objective Function
A_ub: inequality constraints coefficients but in the form of <=
b_ub: the RHS of those inequality constraints
A_eq: equality constraints coefficients
b_eq: the RHS of those equality constraints

RETURNS:

A: main part of simplex tableau, containing the system of constraints' equations with canonical form
B: RHS of the constraints' equations
C: Coefficients of Objective Function
basic_vars: indices of basic vars in matrix "A"
nonbasic_vars: indices of nonbasic vars in matrix "A"
W: sum of A.Vs vector in terms of original variables of the problem.
AV_indx: indices of A.Vs vars in matrix "A"

—---

# function 2: *rs_1p(C, A, B, basic_vars, nonbasic_vars, C2)*:

revised simplex 1 phase function
it takes its inputs after putting them in std canonical form (aka: output of put_to_std_form function) it does the looping of the revised simplex algorithm till it terminates.

INPUTS:

C: Coefficients of Objective Function
A: main part of simplex tableau, containing the system of constraints' equations with canonical form
B: RHS of the constraints' equations
basic_vars: indices of basic vars in matrix "A"
nonbasic_vars: indices of nonbasic vars in matrix "A"
C2: Secondary Coefficients to manipulate in the process (like original OF when minimizing W)

RETURNS:

A,B,C,basic_vars,C2 like the inputs, but modified
XB: New RHS
new_Ps: New non-basic columns of matrix "A"

—---

# main function: *revised_simplex(c, A_ub=[], b_ub=[], A_eq=[], b_eq=[])*:
The main Revised Simplex function, with 2 phase capability if needed.

INPUTS:
c: coefficients of Objective Function
A_ub: inequality constraints coefficients but in the form of <=
b_ub: the RHS of those inequality constraints
A_eq: equality constraints coefficients
b_eq: the RHS of those equality constraints

RETURNS:
f: optimal function value
basic_vars: the indices of basic variables
XB: the corresponding values to the basic variables, RHS.

—---

You can find the detailed commented code attached with this report.

5. Comparison with ***Python's Scipy*** built-in function

I did compare my code to the built-in function of python Scipy. The results were the same, however there was a time difference between the two. The built-in function was more efficient by a double factor. I suspect that this is because of the printing I do in my code, plus of course they are doing it in a more efficient optimized manner. You can find the detailed comparisons with the time taken in the attached notebook. I tried to test on 3 test cases I obtained from the last assignment.

6. Applications

As said in the introduction, the LPP and the revised simplex method of solving it can be used in many real case operational research problems in the industry. I added the problem we used to solve in the slides to the end of the notebook (Interior and Exterior paint production).