Linear and Nonlinear Programming
- Optimization -

Revised Simplex Method
for solving LLP

Supervised by:
Dr. Ahmed AbdelSamea

MATH 404
Fall 2022

Submitted by:
Abdullah Kamal Muhammad
201801271

INDEX

S.no **CONTENTS**

MOTIVATION

While modelling any big system to be described by using a linear optimization problem, it may result in very long equations that contain many variables and many constraints. The traditional Simplex method store all variables to reach the optimal solution although only few variables are used to reach it and it others have no contribution. To solve that problem, it will be more powerful to perform these long mathematical and matrices operation on a powerful computer that have a powerful computational capacity. Unfortunately, the size of the modeled system may big bigger than the capacity of the storage on the digital devices due to huge number of calculations and variable that need to be saved from one iteration to another. New method called "Revised Simplex Method" was developed to avoid that problem and to store only the needed information to iterate from one stage to another to find the optimal solution. Revised Simplex Method only transforms the elements of an inverse matrix. It assumes a new principle and structure called the "Decomposition Principle". By applying this principle we can also trace the changes over the variables through different stages until it reaches the values that result in the optimal solution. Revised Simplex deals with a specific part form the tableau that we be used.

THEORY

Revised Simplex Method avoids storing the entire tableau to deal with the big Systems, but alternatively it stores just important variables to deal with such as:

- The relative cost coefficients Cs which are used to determine which variable Xs will enter the basic variable set in the next iteration.

$$\overline{c}_s = \min(\overline{c}_j)$$

- If there is at least one negative coefficient so there is an ability to improve the solution with more iterations.

  The entering variable will be added to the tableau with new form to be used.

- The entering variable has to enforce one variable Xr to leave the basic variable set.

  To determine the leaving variable, we find ratio between the entering entities values and the corresponding values of the solutions Xb. We choose the minimum positive value and determine the corresponding variable to leave.

$$\overline{\mathbf{A}}_s = \begin{Bmatrix} \bar{a}_{1s} \\ \bar{a}_{2s} \\ \vdots \\ \bar{a}_{ms} \end{Bmatrix} \quad \overline{\mathbf{X}}_B = \begin{Bmatrix} \bar{b}_1 \\ \bar{b}_2 \\ \vdots \\ \bar{b}_m \end{Bmatrix}$$

$$\frac{\bar{b}_r}{\bar{a}_{rs}} = \min_{\bar{a}_{is} > 0} \left\{ \frac{\bar{b}_i}{\bar{a}_{is}} \right\}$$

With the new tableau Revised Simlex method uses the inverse basis matrix in Order to find the needed variables and quantities until it finds the optimial solution.

For simiplicity, it is considered that the Revised Simplex method starts with the second phase with a feasible solution to use. And it is applicable after that to start with phase one or two.

It works on the objective function which will be included into the tableau calcultation and its form will contain vcoeffiecients for all variable in LP.

$$f(\mathbf{X}) = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

subject to

$$\mathbf{AX} = \mathbf{A}_1x_1 + \mathbf{A}_2x_2 + \cdots + \mathbf{A}_nx_n = \mathbf{b}$$

$$\underset{n\times 1}{\mathbf{X}} \geq \underset{n\times 1}{\mathbf{0}}$$

From the given problem, we extract the needed vectors and matrices to initiate the the tableau. We need A, B, C matrices. A is the coefficients matrix of the constraints functions, B is the solution of the constraints functions which will be Xb represented in the upcoming steps and C is the coefficients of the objective function.

Assuming that the linear programming problem has a solution, let

$$\mathbf{B} = [\mathbf{A}_{j1}\ \mathbf{A}_{j2}\ \cdots\ \mathbf{A}_{jm}]$$

be a basis matrix with

$$\underset{m\times 1}{\mathbf{X}_B} = \begin{Bmatrix} x_{j1} \\ x_{j2} \\ \vdots \\ x_{jm} \end{Bmatrix} \quad \text{and} \quad \underset{m\times 1}{\mathbf{c}_B} = \begin{Bmatrix} c_{j1} \\ c_{j2} \\ \vdots \\ c_{jm} \end{Bmatrix}$$

With each new iteration we need to update the vector Xb with the corresponding Basic variable set.

$$\mathbf{X}_B = \mathbf{B}^{-1}\mathbf{b} = \overline{\mathbf{b}} \geq \mathbf{0}$$

Revised Simplex method use the Objective function into the tableau to set its coefficient as the same procedure with the constraint functions.

$$\sum_{j=1}^{n} \mathbf{P}_j x_j + \mathbf{P}_{n+1}(-f) = \mathbf{q}$$

Building the inverse matrix:

We define D matrix as

$$\begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{c}_B^{\mathrm{T}} & 1 \end{bmatrix}$$

And it is easier to split D into 4 small matrices and find their inverses then

Combine them again to find the inverse of D matrix as a whole.

$$\mathbf{D}^{-1} = \begin{bmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}_B^{\mathrm{T}}\mathbf{B}^{-1} & 1 \end{bmatrix}$$

We will get an new updated form such as,

$$\begin{array}{llllll} x_{j1} & & & & & \bar{b}_1 \\ \quad x_{j2} & & & & & \bar{b}_2 \\ \qquad \vdots & + & \sum\limits_{j\,\text{nonbasic}} & \overline{\mathbf{A}}_j x_j & = & \vdots \\ \qquad x_{jm} & & & & & b_m \\ \qquad -f & + & \sum\limits_{j\,\text{nonbasic}} & \bar{c}_j x_j & = & -f_0 \end{array}$$

As mentioned with each iteration, the entering variables take an update form

on the previous values with the effect of B inverse matric:

$$\overline{\mathbf{A}}_j = \mathbf{B}^{-1}\mathbf{A}_j$$

The same procedure of entering and leaving new variable lasts until the method Finds the optimum solution or by checking the special cases of the LP.

The steps of

- finding a feasible solution to start with
- determining the entering variable with the most negative value
- determining the entering variable with the minimum positive value
- find a new form of the entering variable after a multiplication with the inverse matrix exists.
- Formulate an update on the tableau
- Repeating the previous step until it finds all values of the objective function are positive so that the current solution is the optimum or by checking the special cases.

ALGORITHM:

According to Engineering Optimization Theory textbook, Revised Simplex method can work on phase 1 or phase 2, but for simplicity it finds its formula firstly based on phase 2.

- To start the algorithm it has to take the LP in the canonical form with all needed

  variables, it does not matter the type of the variable; basic, slack or artificial

$$\begin{array}{llll}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + x_{n+1} & & & \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & +x_{n+2} & & \\
\vdots & & & \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & & +x_{n+m} & \\
c_1x_1 + c_2x_2 + \cdots + c_nx_n & & & -f
\end{array}$$

- The algorithm starts the iteration by filling the tableau with the coefficient of

  the basic variables and the constants or the results vector

  it will result in B matrix with the form of the Identity matrix at the first iteration

  so that the inverse of B is the same matrix Identity.

$$\mathbf{B} = \mathbf{I}, \text{ and its inverse } \mathbf{B}^{-1} = [\beta_{ij}]$$

- The relative costs are computed to determine the entering variable

  Note that if we start with phase one we need to work with -w- function

  to find a feasible solution to start phase 2

- We look for the most negative value among the negative values we found form the previous step and the corresponding variable will enter the basic set.
- For the entering variable, we have to find its new values respected to B invers Matrix by multiplying B inverse and the entering variable

$$\overline{\mathbf{A}}_s = \mathbf{B}^{-1}\mathbf{A}_s = \overline{\beta}_{ij}\mathbf{A}_s$$

$$\begin{aligned}
\overline{a}_{1s} &= \beta_{11}a_{1s} + \beta_{12}a_{2s} + \cdots + \beta_{1m}a_{ms} \\
\overline{a}_{2s} &= \beta_{21}a_{1s} + \beta_{22}a_{2s} + \cdots + \beta_{2m}a_{ms} \\
&\vdots \\
\overline{a}_{ms} &= \beta_{m1}a_{1s} + \beta_{m2}a_{2s} + \cdots + \beta_{mm}a_{ms}
\end{aligned}$$

- Here can perform a check on the value of the entering vector for any special case
- Then we find the Ratio vector by dividing Xb on As

  Or the decision of result vector on the entering vector
- We choose the minimum positive value to let the corresponding variable to Leave the basic set to be replaced by the entering variable.

$$\frac{\overline{b}_r}{\overline{a}_{rs}} = \min_{\overline{a}_{is} > 0} (\overline{b}_i / \overline{a}_{is})$$

In the case of a tie, choose r at random.

- After filling the tableau with the new values with the entering variable and

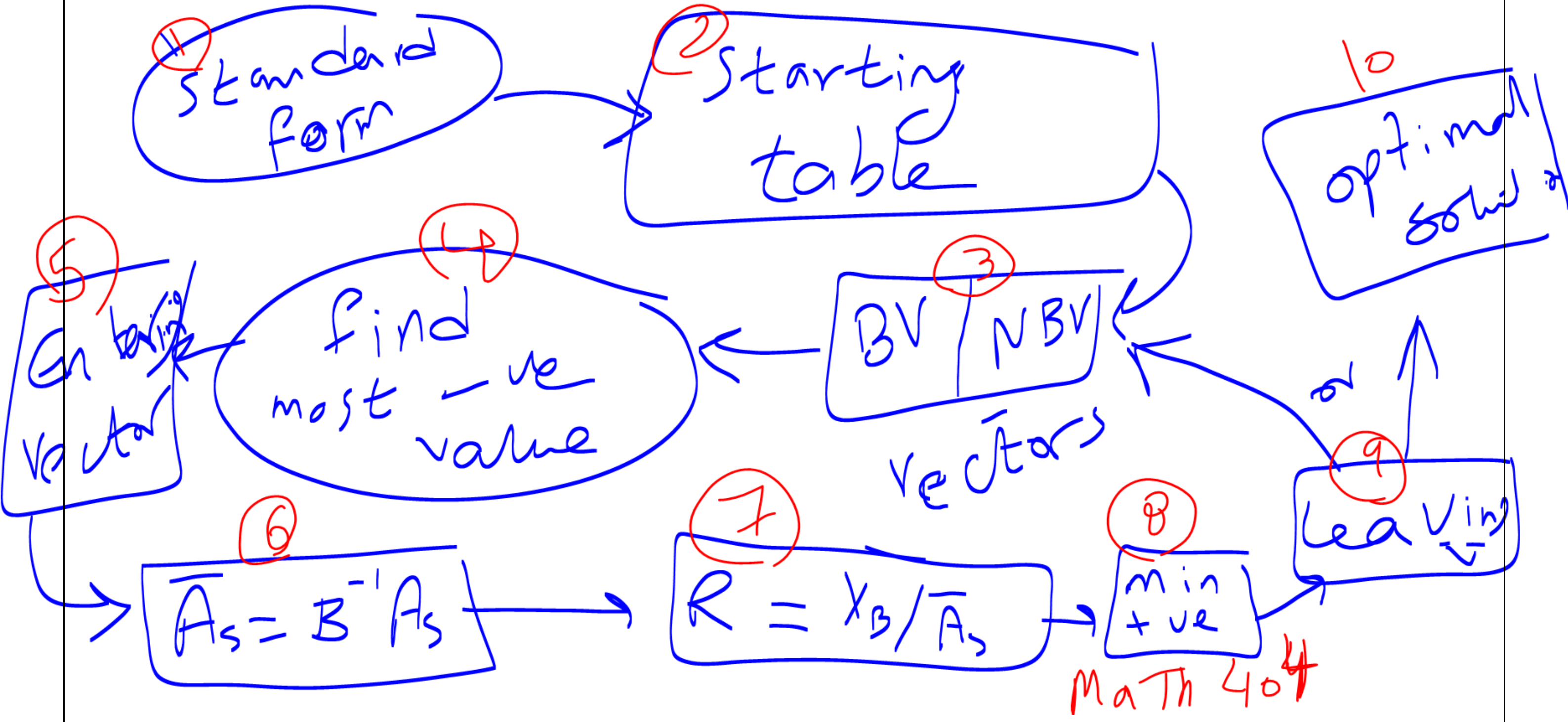
  B inverse matrix, we check if we have found the optimum solution:

  -if we find that all objective function values have become positive

  - all values of the Ratio column are negative then we check the special cases like unbounded problem for example.

- If we did not achieve the optimum solution, we repeat the steps again

  Starting from the step of finding the relative costs.



Implementation

The implementation as mentioned previously takes the canonical form into a matrices A, B, C
And it needs to know the basic variable to set the Basic set for the first iteration.

I have use the example explained in the textbook as a reference to run my code

Example 4.1

$$\text{Maximize } F = x_1 + 2x_2 + x_3$$

subject to

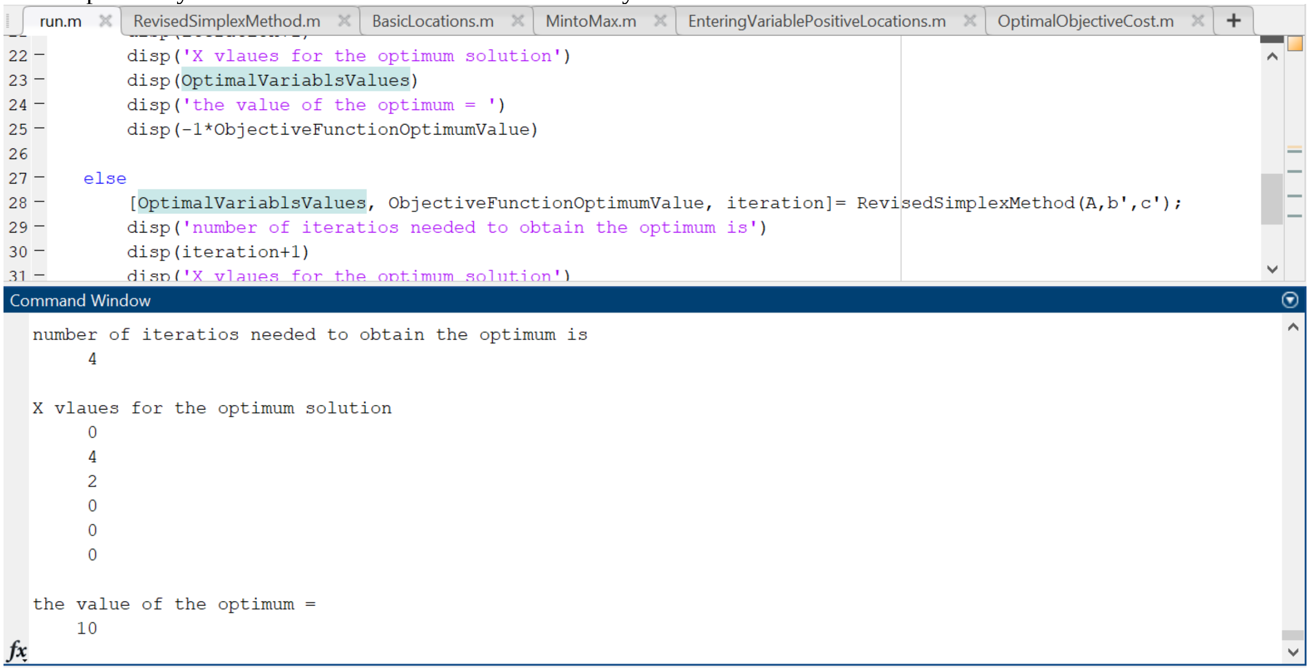
$$2x_1 + x_2 - x_3 \le 2$$
$$-2x_1 + x_2 - 5x_3 \ge -6$$

$$4x_1 + x_2 + x_3 \le 6$$
$$x_1 \ge 0,\ \ x_2 \ge 0,\ \ x_3 \ge 0$$

$$\begin{aligned}
2x_1 + x_2 - x_3 + x_4 \quad & = 2 \\
2x_1 - x_2 + 5x_3 \quad + x_5 \quad & = 6 \\
4x_1 + x_2 + x_3 \quad\quad + x_6 & = 6 \\
x_i \ge 0, \quad i = 1 \text{ to } 6 &
\end{aligned}$$

The implementation code is attached at the end of the file. I followed the 3rd reference to implement it.

The output of my code is the same as the solution found by the textbook



```
        disp('X vlaues for the optimum solution')
        disp(OptimalVariablsValues)
        disp('the value of the optimum = ')
        disp(-1*ObjectiveFunctionOptimumValue)

    else
        [OptimalVariablsValues, ObjectiveFunctionOptimumValue, iteration]= RevisedSimplexMethod(A,b',c');
        disp('number of iteratios needed to obtain the optimum is')
        disp(iteration+1)
        disp('X vlaues for the optimum solution')
```

```
number of iteratios needed to obtain the optimum is
     4

X vlaues for the optimum solution
     0
     4
     2
     0
     0
     0

the value of the optimum =
    10
```

Application

In terms of time spent and eliminating the usage of sparse matrices, the improved simplex method is a more effective way to solve LP problems.

Due to the efficient approach in solving the LP, there are many applications in which Revised Simplex method is used, such as

- Sensitivity Analysis
- Many planning companies and factories use it to ensure optimal production planning [3].
- One fundamental challenge of manufacturers is how to maximize profit while keeping standards for a product mix.
- The paper I use a reference is working on Revised Simplex to choose between 15 different types of paints to maximize its profits.

Comparison

It was performed with the given example of the textbook.

$f_{\min} = -10$

[illegible] $F_{max} = 10$
as The output of my Code

References

[1]

[1]“Engineering Optimization: Theory and Practice: Rao, Singiresu S.: 9780470183526: Amazon.com: Books,” *Amazon.com*, 2022. https://www.amazon.com/Engineering-Optimization-Practice-Singiresu-Hardcover/dp/B010WFO8HI (accessed Nov. 27, 2022).

[2] “Application Of Revised Simplex Method For Profit Maximization In A Paint Company,” *www.researchjournali.com*. https://researchjournali.com/view.php?id=5826 (accessed Nov. 27, 2022).

[3]“Revised Simplex Method (Introduction, Steps and Example),” *BYJUS*. https://byjus.com/maths/revised-simplex-method/

```
% matrices and vectors in the standard form:
% the user should enter like that given example
% A martix is the Coefficients of the constraints
% b is the right hand part of these Coefficients
% A martix is the Coefficients of the Objective function

%A = [2 1 -1 1 0 0;2 -1 5 0 1 0;4 1 1 0 0 1];
%b = [2 6 6]';
%c = [1 2 1 0 0 0]';
                       % the initial basic feasible solution
A = input('Enter matrix A: ');
b = input('Enter vector of b: ');
c = input('Enter vector of c: ');
typeofLP = input('Enter   1   Maximzmiation or   2     minimization:
 ');
disp('-----------------------
Solution---------------------------------')

if typeofLP == 1
    c = MintoMax(c)
    [OptimalVariablsValues, ObjectiveFunctionOptimumValue, iteration]=
 RevisedSimplexMethod(A,b',c');
    disp('number of iteratios needed to obtain the optimum is')
    disp(iteration+1)
    disp('X vlaues for the optimum solution')
    disp(OptimalVariablsValues)
    disp('the value of the optimum = ')
    disp(-1*ObjectiveFunctionOptimumValue)

else
    [OptimalVariablsValues, ObjectiveFunctionOptimumValue, iteration]=
 RevisedSimplexMethod(A,b',c');
    disp('number of iteratios needed to obtain the optimum is')
    disp(iteration+1)
    disp('X vlaues for the optimum solution')
    disp(OptimalVariablsValues)
    disp('the value of the optimum = ')
    disp(ObjectiveFunctionOptimumValue)


end
```

Published with MATLAB® R2020b

```
function [OptimalVariablsValues, ObjectiveFunctionOptimumValue,
 iteration] = RevisedSimplexMethod(A,b,c)
   % XB is the final values for the variables
   % From XB we will know the optimal values needed to make
   % the objective function optimum also
   XB = zeros(length(c),1);

   % I need to know the location of Basic variable to fill the
 tableau
   % I use the function BasicLocations to find them it takes the
   % coefficients of the objective function and any variable = 0 in
 the
   % first iteration is a basic for that iteration
   Basic_Locations = [];
   Basic_Locations = BasicLocations(c);

   % Create the initial Basis matrix, compute its inverse and
   % compute the inital basic feasible solution
   %------------------------------------------
   %Basic  |  B inverse | Xb
   %that is the shape I am using

   % coefficient mar
   B=A(:,Basic_Locations);
   B_inverse = inv(B);
   XB(Basic_Locations) = B_inverse*b;


   % Iteration phase
   iteration = 0;
   while iteration < 15

       % Compute the vector of reduced costs c_r

       c_B = c(Basic_Locations);         % Basic variable costs
       p = (c_B'*B_inverse)';   % Dual variables
       c_r = c' - p'*A;    % Vector of reduced costs

       % Check if the solution is optimal. If optimal, use
       % 'return' to break from the function, e.g.

       %locations of any negative costs which I chose the most one to
       %determine the corresponding  entering variable

       NegativeCostsLocations = OptimalObjectiveCost(c_r);
       if (length(NegativeCostsLocations) == 0)

       %it enter if there are no negative costs and here we reached
 the
       %optimumm solution and there is no any improvement may happen

           ObjectiveFunctionOptimumValue = c'*XB;
```

```
            OptimalVariablsValues = XB;
            return;
        end

        % if there is any negative cost it will perform additional
        % iteration to improve the solution


        %selecting the entering variable
        entering_variable = NegativeCostsLocations(1);

        % as I mentioned in the report, I need to find the new shape
 of
        % the entering variable by multiplying it to the B inverse
 matrix

        new_entering_variable = B_inverse*A(:,entering_variable);

        %Variable Positive Locations by using the function of
 EnteringVariablePositiveLocations
        % to know if there is any positive values to choose the
 minimum
        % positive one to let the corresponding variable leaves the
 basic
        % set
        PositiveCostsLocations =
 EnteringVariablePositiveLocations(new_entering_variable);
        if (length(PositiveCostsLocations) == 0)
           % it enters here if it does not find any positive value in
 the
           % entering variable which means that I can not let any
 variable
           % in the basic set leaves the Basic set then I can not
 improve the solution

           ObjectiveFunctionOptimumValue = -inf;  % Optimal objective
 function cost = -inf
           OptimalVariablsValues = [];
           % at that situation it should terminate the iteration and
 end
           % the iteratation of improving the solution
           return
        end

        % the minimum postitiv ratio
        RatioColumn =  XB(Basic_Locations(PositiveCostsLocations))./
new_entering_variable(PositiveCostsLocations);
        ChoosingMinmumProstiveValue = min(RatioColumn);

        L = find(XB(Basic_Locations)./new_entering_variable ==
 ChoosingMinmumProstiveValue); % indices with ratio

        % Select the leaving variable
        LeavingVariable = L(1);
        XB(Basic_Locations) = XB(Basic_Locations) -
ChoosingMinmumProstiveValue*new_entering_variable;
        XB(entering_variable) = ChoosingMinmumProstiveValue;


        % forming the basic set by replacing the leaving variable with
the
        % entering variable to start the following iteration

        Basic_Locations(LeavingVariable) = entering_variable;

        % I have to normalize the leaving row and to put zeros above
and
        % below the identity by performing matrix operations learnt
from
        % the linear algebra course before
        % after that I can forrmulate a new structure of B inverse
matrix
        % by concatiniating the entering column to B inverse matrix

        new_tableau=horzcat(new_entering_variable, B_inverse);
        %disp(new_tableau)
        [rows columns]=size(new_tableau);
        if (ChoosingMinmumProstiveValue~=0)

new_tableau(LeavingVariable,:)=new_tableau(LeavingVariable,:)/
new_tableau(LeavingVariable,1);
        % the meaning of a variable is leaving the basic set is a
        % normalization of the corresponding variable in the XB column
and
        % divide the whole row with that value
        end
        % after normalization of the leaving row, I should put zeros
above
        % and below that row by subtaracting a multiple of it from
other
        % rows to formulate the new tableau
        for i = 1:rows % For all matrix rows
           if (i ~= LeavingVariable)
               % I skip the normalized row just to subtract from the
above
               % and below rows
               new_tableau(i,:)=new_tableau(i,:)-
new_tableau(i,1)*new_tableau(LeavingVariable,:);
               % that result in the new tableau after subtracting the
               % multiple of the normalized to get zeros above and
below
        end
        end
        B_inverse=new_tableau(1:3,2:end);
        iteration = iteration + 1;
    end
    if iteration > 15
```

```
        disp('Max number of iterations performed!');
    end
    return
end
```

Published with MATLAB® R2020b

```
function listOFBasic = BasicLocations(C)
    listOFBasic = [];
    for i = 1: length(C)
        if C(i) == 0
            listOFBasic(end+1) = i;
        end
    end
    %disp(listOFBasic);
end
```

Published with MATLAB® R2020b

```
function counterOfPositive = EnteringVariablePositiveLocations(Costs)
    counterOfPositive = [];
    for i = 1: length(Costs)
        if Costs(i) > 0
            counterOfPositive(end+1) = i;
        end
    end
end
```

Published with MATLAB® R2020b

```
function newC = MintoMax(C)
    newC = []
    for i = 1: length(C)
        newC(end+1) = -1 * C(i);
    end
end
```

Published with MATLAB® R2020b

```
function counterOfNegative = OptimalObjectiveCost(Costs)
    counterOfNegative = [];
    for i = 1: length(Costs)
        if Costs(i) < 0
            counterOfNegative(end+1) = i;
        end
    end
end
```

Published with MATLAB® R2020b