Dhaka Univ. J. Sci. 60(2): 163-168, 2012 (July)

A Computer Technique for Solving LP Problems with Bounded Variables

S. M. Atiqur Rahman Chowdhury* **and Sanwar Uddin Ahmad**

Department of Mathematics; University of Dhaka, Dhaka-1000, Bangladesh

Received on 27.02.2011. Accepted for Publication on 15. 12.11

Abstract

Linear Programming problem (LPP)s with upper bounded variables can be solved using the Bounded Simplex method (BSM), without the explicit consideration of the upper bounded constraints. The upper bounded constraints are considered implicitly in this method which reduced the size of the basis matrix significantly. In this paper, we have developed *MATHEMATICA* codes for solving such problems. A complete algorithm of the program with the help of a numerical example has been provided. Finally a comparison with the built-in code has been made for showing the efficiency of the developed code.

Keywords: LPP, bounded simplex method, bounded variable, computer program, restrictions.

I. Introduction

Linear Programming problem (LPP) with bounded variables has the form

$$Maximize, Z = \{CX: (A, I)\ X = b; \quad L \le X \le U\} \qquad (1)$$

One can solve such problems by regular simplex method by considering the lower and upper bound constraints explicitly which is not computationally efficient as the number of constraints as well as the number of variables become very large.

Dantzig[1] developed the method for solving linear programming with upper bound restrictions on the variables. Wagner[2] developed the dual simplex method for LPP with bounded variables, which is further studied by Maros[3, 4]. In 1972, Duguay[5] et al. studied linear programming with relative bounded variables. Y. Xia and J. Wang[6] used neural network for finding an approximate solution of LPP with bounded variables which converges globally to the solutions to the LPPs.

However, most of the real life problems deal with large quantities of data and parameters, which can only reasonably be tackled by the calculating power of a computer and researchers, are actively engaged in developing programming codes for solving various problems in Operations Research e.g. Saha and Hasan[7], Hasan.[8], Morshed and Hasan[9].

In this paper, we will discuss the algorithm for solving LPP with Bounded Variables and develop a MATHEMATICA program for solving such problems, which is capable of dealing with any number of variables. Finally a numerical example obtained by using the program is provided and a comparison has been made with the built-in MATHEMATICA code.

II. Bounded Simplex Method[10]

In LP models variables may have explicit positive upper and lower bounds. An LPP may have in addition to the regular constraints, lower or upper bounds on some or all variables i.e. constraints of the type

$$L \le X \le U$$

For the lower bounds i.e. $X \ge L$ in (1), the following substitution

$$X - X' = L, \quad X' \ge 0 \qquad (2)$$

can be used throughout and solve the problem in terms of X'(whose lower bound now equals zero). The original X is determined by back substitution which is legitimate because its guarantees that $X - X' = L$ will remain nonnegative for all $X' \ge 0$.

However, for the upper bounding constraint, $X \le U$, the idea of direct substitution i.e.

$$X + X'' = U, X'' \ge 0$$

is not correct because back substitution,

$X + X'' = U$, does not ensure that X will remain nonnegative.

For simplicity we define the upper bounded LP model as

$$Maximize, Z = \{CX: (A, I)\ X = b, \quad 0 \le X \le U\} \qquad (3)$$

The bounded algorithm uses only the constraints $(A, I)\ X = b, X \ge 0$ explicitly, while accounting for $X \le U$ implicitly through modification of the simplex feasibility condition.

Let $X_B = B^{-1}b$ be a current basic feasible solution of $(A, I)\ X = b, X \ge 0$ and suppose that according to the (regular) optimality condition, P_j is the entering vector. Then, given that all the non-basic variables are zero, the constraint equation of the i^{th} basic variable can be written as

$$(X_B)_i = (B^{-1}b)_i - \left(B^{-1}P_j\right)_i x_j$$

* Correspondence author : E-mail: Chowdhury_sher@yahoo.com

When the entering variable x_j increases above zero level, $(X_B)_i$ will increase or decrease depending on whether $\left(B^{-1}P_j\right)_i$ is negative or positive, respectively. Thus, in determining the value of the entering variable x_j three conditions must be satisfied:

Condition1: The basic variable $(X_B)_i$ remains non-negative-that is, $(X_B)_i \geq 0$.

This is not violated if

$$x_j \leq \theta_1 \equiv \min_i \left\{ \frac{(B^{-1}b)_i}{(B^{-1}P_j)_i} : \left(B^{-1}P_j\right)_i > 0 \right\} \quad (4)$$

Condition2: The basic variable $(X_B)_i$ does not exceed its upper bound i.e., $(X_B)_i \leq (U_B)_i$, where U_B comprise the ordered elements of U corresponding to X_B.

This is not violated if

$$x_j \leq \theta_2 \equiv \min_i \left\{ \frac{(B^{-1}b)_i - (U_B)_i}{(B^{-1}P_j)_i} : \left(B^{-1}P_j\right)_i < 0 \right\} \quad (5)$$

Condition3: The entering variable x_j cannot assume a value larger than its upper bound- that is $x_j \leq u_j$, where u_j is the jth element of U.

Combining the three restrictions together, x_j enters the solution at the level that satisfies all three conditions and the level is defined as

$$x_j = min\{\theta_1, \theta_2, u_j\} \quad (6)$$

The change of basis for the next iteration depends on whether x_j enters the solution at level θ_1, θ_2 or u_j. Assuming that $(X_B)_r$ is the leaving variable, then we have the following rules:

Rule1: $x_j = u_j$. The basic vector X_B remains unchanged because $x_j = u_j$ stops short of forcing any of the current basic variables to reach its lower (=0) or upper bound. This means that x_j will remain nonbasic but at upper bound. Following the argument just presented, the new iteration is generated by using the substitution $x_j = u_j - x_j'$.

Rule2: $x_j = \theta_1$; $(X_B)_r$ leaves the basic solution (becomes non-basic) at level zero. The new iteration is generated in the normal simplex manner by using x_j and $(X_B)_r$ as the entering and the leaving variables, respectively.

Rule3: $x_j = \theta_2$; $(X_B)_r$, becomes non-basic at its upper bound. The new iterations is generated as in the case of, $x_j = \theta_1$, with one modification that accounts for the fact that $(X_B)_r$ will be nonbasic at upper bound. Because the values of θ_1 and θ_2 are developed under the assumption that all nonbasic variables are at zero level. This is achieved by using the substitution $(X_B)_r = (U_B)_r - (X_B)'_r$, where $(X_B)'_r \geq 0$. It is immaterial whether the substitution is made before or after the new basis is computed.

A tie among θ_1, θ_2, and u_j may be taken arbitrarily. However, it is preferable to implement the rule for $x_j = u_j$ because it entails less computation.

The substitution $x_j = u_j - x_j'$ will change the original c_j, P_j and b to $c_j' = -c_j, P_j' = P_j$ and $b\text{-}to\text{-}b' = -u_j P_j$. This means that if the revised simplex method is used, all the computation (e.g, $B^{-1}, X_B,$ and $z_j - c_j$) should be based on the updated values of $C, A,$ and b at each iteration.

III. MATHEMATICA codes

For convenience we have presented the developed MATHEMATICA codes for solving LPP with bounded variables while solving a numerical example.

LPP1

Maximize, $\boldsymbol{Z = 3x_1 + 5x_2 + 2x_3}$

Subject to, $\boldsymbol{x_1 + 2x_2 + 2x_3 \leq 10}$

$$\boldsymbol{2x_1 + 4x_2 + 3x_3 \leq 15}$$

$$\boldsymbol{0 \leq x_1 \leq 4, 0 \leq x_2 \leq 3, 0 \leq x_3 \leq 3}$$

Introducing slack variables $x_4 \geq 0$, $x_5 \geq 0$ one to each constraint, we have

Maximize, $\boldsymbol{Z = 3x_1 + 5x_2 + 2x_3 + 0x_4 + 0x_5}$

Subject to,

$$\boldsymbol{x_1 + 2x_2 + 2x_3 + x_4 + 0x_5 = 10}$$

$$\boldsymbol{2x_1 + 4x_2 + 3x_3 + 0x_4 + x_5 = 15}$$

$$\boldsymbol{0 \leq x_1 \leq 4, 0 \leq x_2 \leq 3, 0 \leq x_3 \leq 3, x_4 \geq 0, x_5 \geq 0}$$

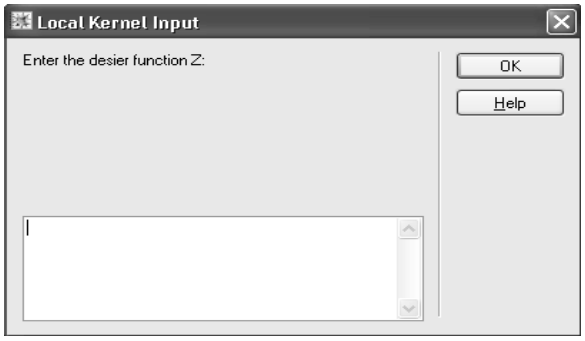
The following codes are used to take the necessary inputs.

```
In[1]:= Exit[]

In[1]:= Off[General::spell]; Off[Set::write]; Off[Table::iterb];
        << LinearAlgebra`MatrixManipulation`
```

```
Zmax = Input["Enter the desier function Z:"];
n = Input["Enter no. of terms n:"]; m = 2; b1 = 2;
Do[x[i] = Input["Enter Row1:"], {i, 1, n}];
lst1 = Table[x[i], {i, 1, n}];
Do[y[i] = Input["Enter Row2:"], {i, 1, n}];
lst2 = Table[y[i], {i, 1, n}];
Do[z[i] = Input["Enter Coeff. of Objective function:"],
 {i, 1, n}]; lst4 = Table[z[i], {i, 1, n}];
Do[U[i] = Input["Enter Upperbound:"], {i, 1, n}];
lst5 = Table[U[i], {i, 1, n}];
Do[B[i] = Input["Enter Coeff. of Basis elements:"],
 {i, 1, m}]; lst6 = Table[B[i], {i, 1, m}];
Do[CT[i] = Input["Enter Constant:"], {i, 1, b1}];
lst7 = Table[CT[i], {i, 1, b1}]; lst8 = {x1, x2, x3, x4, x5};
BV = {x4, x5}; x1 = Input["Enter x1:"];
x2 = Input["Enter x2:"]; x3 = Input["Enter x3:"];
x4 = Input["Enter x4:"];
x5 = Input["Enter x5:"];
```

The inputs are made into the following input box:



Enter the desire function Z:
$\mathbf{3}x_1 + \mathbf{5}x_2 + \mathbf{2}x_3$;
Then, n = 5;
Enter Row 1: 1, 2, 2, 1, and 0
Enter Row 2: 2, 4, 3, 0, and 1
Enter co-eff. of objective function: 3, 5, 2, 0, and 0
Enter upper bound: 4, 3, 3, 9999, and 9999.
Enter co-eff. of basis element: 0, 0
Enter constant: 10, 15
$X_1 = 0; X_2 = 0; X_3 = 0;\ X_4 = 0;\ X_5 = 0$
The initial simplex table is generated by the following codes.

```
Block[{}, a = Normal[SparseArray[lst1, n]];
  b = Normal[SparseArray[lst2, n]];
  h = {a, b};
  l1 = Length[h[[1]]];
  l2 = l1;
  Do[temp[i] = TakeColumns[h, {i}], {i, 1, l1}];
  k = Table[temp[i], {i, 1, l1}];
  CJ = Normal[SparseArray[lst4, n]];
U = Normal[SparseArray[lst5, n]];
  o2 = Table[temp[i] = U[[i]], {i, 1, Length[U]}];
B = Normal[SparseArray[lst6, m]];
CNT = Normal[SparseArray[lst7, b1]];
  Do[{temp[l1 + 1] = CJ[[i]], l1 = l1 + 1}, {i, 1, Length[CJ]}];
l3 = l1; m = Table[temp[i], {i, 1, l3}];
  g = Table[temp[i] = B[[i]], {i, 1, Length[B]}];
  s = Table[temp[i] = CNT[[i]], {i, 1, Length[CNT]}]];
```

Here, a, b are the 1st and 2nd row of the initial table, K is the coefficient matrix, U[[i]] represents the upper bounds of x_i, CNT is the initial basic feasible solutions (b_i), B is the basis matrix and CJ[[i]] represents the co-efficient of the profit vector (C_i).

Table. 1. Initial simplex table

C_j		3	5	2	0	0	Solution(b_i)
Basis		x_1	x_2	x_3	x_4	x_5	
0	x_4	1	2	2	1	0	10
0	x_5	2	(4)	3	0	1	15
$\bar{C}_j$		3	5	2	0	0	Z = 0

The following codes are used for identifying the entering non-basic variable.

```
1 st Iteration :
   Label["at"];
l4 = CJ - g.h; MP = Max[Cases[l4, x_ /; x > 0]];
V = Position[l4, MP];
g5 = Flatten[Table[temp[i] = V[[i]], {i, 1, Length[V]}]];
y = g5[[1]]; θ2 = 10000000;
w = If[y == g5[[1]], k[[y]], Print["Infeasible"]];
θ3 = (U[[y]] - CNT[[y]])/((-1) * w[[y]]); {{d1}, {d2}} = CNT / w;
θ1 = Min[Cases[{d1, d2}, x_ /; x > 0]]; {k6} = w[[1]] * w[[2]];
k7 = If[k6 > 0, θ2, θ3];
θ = Min[θ1, k7, U[[y]]];
```

Here, l4 = $\bar{C}_j = C_j - z_j$ = relative profit vector, MP = $Max\{\bar{C}_j : \bar{C}_j > 0\}$, θ_3 is the value of θ_2 in (5), θ_1 is the value of θ_1 in (4) and θ represents the value of x_j in (6).

By Rule1 in section 2, we obtained Table 2 as,

Table. 2.

Basic	x_1	x'_2	x_3	Solution
Z	3	–5	2	15
x_4	1	− 2	2	4
⇨ x_5	(2)	−4	3	3

The developed codes for Rule1 are,

```
2 nd Iteration :
  If[θ == U[[y]], {k[[y]] = -k[[y]], CJ[[y]] = -CJ[[y]],
    CNT[[1]] = CNT[[1]] - U[[y]] * a[[y]],
    CNT[[2]] = CNT[[2]] - U[[y]] * b[[y]], a[[y]] = -a[[y]],
    b[[y]] = -b[[y]], h = {a, b}, BV[[1]] = CNT[[1]],
    BV[[2]] = CNT[[2]]}]; {{t1}} = Position[{d1, d2}, θ1];
t2 = If[t1 > 1, h[[1]], h[[2]]];
t3 = If[t1 > 1, CNT[[1]], CNT[[2]]]; l4 = CJ - g.h;
MP = Max[Cases[l4, x_ /; x > 0]]; V = Position[l4, MP];
g5 = Flatten[Table[temp[i] = V[[i]], {i, 1, Length[V]}]];
y = g5[[1]]; θ2 = 10000000;
w = If[y == g5[[1]], k[[y]], Print["Infeasible"]];
θ3 = (U[[y]] - CNT[[y]])/((-1) * w[[y]]); {{d1}, {d2}} = CNT / w;
θ1 = Min[Cases[{d1, d2}, x_ /; x > 0]]; {k6} = w[[1]] * w[[2]];
k7 = If[k6 > 0, θ2, θ3]; θ = Min[θ1, k7, U[[y]]];
If[k == h, Goto["at"]];
{{r1}} = Position[{θ1, k7, U[[y]]}, θ1];
```

Table 3 is obtained by rule 2 in section 2 as,

Table. 3.

Basic	x'_2	x_3	x_5	Solution
Z	1	–5/2	−3/2	39/2
x_4	0	1/2	-1/2	5/2
⇨ x_1	(−2)	3/2	1/2	3/2

The developed codes for Rule2 are,

```
3 rd Iteration :
  If[θ == θ1, {lst8[[y]] = BV[[t1]],
    h = {t4 = t2 - h[[t1]] / b[[y]], t5 = h[[t1]] / b[[y]]},
    CNT[[1]] = t3 - CNT[[t1]] / b[[y]],
    CNT[[2]] = CNT[[t1]] / b[[y]], g[[t1]] = CJ[[y]],
    l4 = CJ - g.h, MP = Max[Cases[l4, x_ /; x > 0]],
    V = Position[l4, MP],
    g5 = Flatten[Table[temp[i] = V[[i]], {i, 1, Length[V]}]],
    y = g5[[1]], θ2 = 10000000,
    w = If[y == g5[[1]], {{t4[[y]]}, {t5[[y]]}},
      Print["Stop"]], v = t4[[y]] * t5[[y]],
    If[v == 0, {w1 = {{1}, {1}}}], w3 = If[v == 0, w1, w],
    {{d1}, {d2}} = CNT / w3,
    θ4 = Min[Cases[{d1, d2}, x_ /; x > 0]], If[v == 0, θ4 = θ2],
    θ3 = (U[[r1]] - CNT[[y]])/((-1) * w[[y]]), {k8} = w[[1]] * w[[2]],
    k9 = If[k8 > 0, θ2, θ3], θ = Min[θ4, k9, U[[y]]],
    BV[[1]] = CNT[[1]], BV[[2]] = CNT[[2]],
    lst8[[r1]] = BV[[y]]}];
{{r}} = Position[{θ4, k9, U[[y]]}, k9];
t10 = If[r > 1, h[[1]], h[[2]]];
t11 = If[r > 1, CNT[[1]], CNT[[2]]];
```

For Rule3, developed codes are:

```
Final Iteration and Result :
  If[{θ} == k9, {lst8[[r1]] = lst8[[y]], b = h[[r]], a = t10,
    If[a[[r]] == 0, {t6 = h[[r1]]}, {t6 = t10 - h[[r]] / b[[y]]}],
    h = {t6, t9 = h[[r]] / b[[y]]},
    If[a[[r]] == 0, {CNT[[1]] = CNT[[r1]]},
     {CNT[[1]] = t11 - CNT[[r]] / b[[y]]}],
    CNT[[2]] = CNT[[r]] / b[[y]], g[[r]] = CJ[[y]],
    l4 = CJ - g.h, lst8[[r]] = BV[[r]]}]; a = h[[r1]];
c = h[[r]]; If[Length[Cases[l4, x_ /; x > 0]] > 0, Goto["tt"]];
a[[r1]] = -a[[r1]]; c[[r1]] = -c[[r1]];;
op1 = CNT[[1]] + U[[r1]] * a[[r1]];
op2 = CNT[[2]] + U[[r1]] * c[[r1]]; CJ1 = CJ;
CJ1[[r1]] = -CJ1[[r1]]; {p1, p2} = {a, c}; h1 = {p1, p2};
l5 = CJ1 - g.h1; MP1 = Max[Cases[l5, x_ /; x > 0]];
s = {op1, op2}; w4 = If[MP1 == g5[[1]], h1[[y]], "End"];
lst8[[r1]] = U[[r1]]; lst8[[r]] = U[[r]] - op2; x1 = lst8[[1]];
x2 = lst8[[2]]; x3 = lst8[[3]]; x4 = lst8[[4]]; x5 = lst8[[5]];
If[g[[1]] + g[[2]] == 0,
 Print["infeasible and so program is stop here"],
 {Print["Our program is:\n", w4], "\n",
  Print["Optimal solutions are:\n", "x1" → x1, "\n",
   "x2" → x2, "\n", "x3" → x3], "\n",
  Print["Final optimization table is:\n Cj = ",
   {" "CJ1} // MatrixForm], "\n",
  Print["basis:  Simplex table:  Constant:\n",
   g // MatrixForm, h1 // MatrixForm, s // MatrixForm],
  "\n", Print["Cj-Zj=", {l5} // MatrixForm], "\n",
  Print["Maximum of Objective Function Z is\n", Zmax]}];
```

These codes are iteratively implemented until the optimum table is obtained.

Table. 4.

Basic	x_1	x_3	x_5	Solution
Z	1/2	-7/4	-5/4	75/4
x_4	0	1/2	-1/2	5/2
x'_2	-1/2	-3/4	-1/4	-3/4

Table. 5. Optimum table

Basic	x'_1	x_3	x_5	Solution
Z	-1/2	-7/4	-5/4	83/4
x_4	0	1/2	-1/2	5/2
x'_2	1/2	-3/4	-1/4	5/4

By back substitution the optimal values of x_1, x_2, and x_3 are $x_1 = u_1 - x'_1 = 4$, $x_2 = u_2 - x'_2 = \frac{7}{4}$ and $x_3 = 0$. Finally, the optimal value of Z is $\frac{83}{4}$.

Output

```
Our program is:
End
Optimal solutions are:
x1 → 4
x2 → 7/4
x3 → 0
Final optimization table is:
 Cj = (-3   -5   2   0   0)
basis:  Simplex table:  Constant:
( 0 )  [ 0   0   1/2   1   -1/2 ]  [ 5/2 ]
(-5 )  [ 1/2 1   -3/4  0   -1/4 ]  [ 5/4 ]
Cj-Zj=(-1/2  0   -7/4  0   -5/4 )
Maximum of Objective Function Z is
83/4
```

Table. 6. Comparison of used timed for solving LPP1

Machine Configurations	Time Used(Sec.)	
Intel[R] 4 CPU 2.00GHz,512M B of RAM	Built-in Code	Developed Code
	0.591	0.48

IV. Algorithm of BSM[10, 11]

In this section we present the algorithm of our developed code.

Step 1. Calculate the net evaluation $C_j - Z_j$. For a maximization problem if $C_j - Z_j \leq 0$ for the non-basic variables at their upper bound, optimum basic feasible solution is attained. It not, go to step-2, revenue is true for a minimization problem.

Step 2. Select the most positive $C_j - Z_j$.

Step 3. Let x_j be a non basic variable at zero level which is selected to enter the solution. Compute the quantities defined in (4) & (5)

Step 4. $x_j = min\,(\theta_1, \theta_2, u_j)$.

Sub-step 1: If $\theta = \theta_1.(X_B)_r$ leaves the basic solution (because non-basic) at level zero and x_j enter by using the regular row operation of the simplex method.

Sub-step 2: If $\theta = \theta_2.(X_B)_r$leaves the basic solution at level zero and x_j enters then $(X_B)_r$ being non-basic at its upper bound must be substituted out by using

$$(X_B)_r = (U_B)_r - (X_B)'_r$$

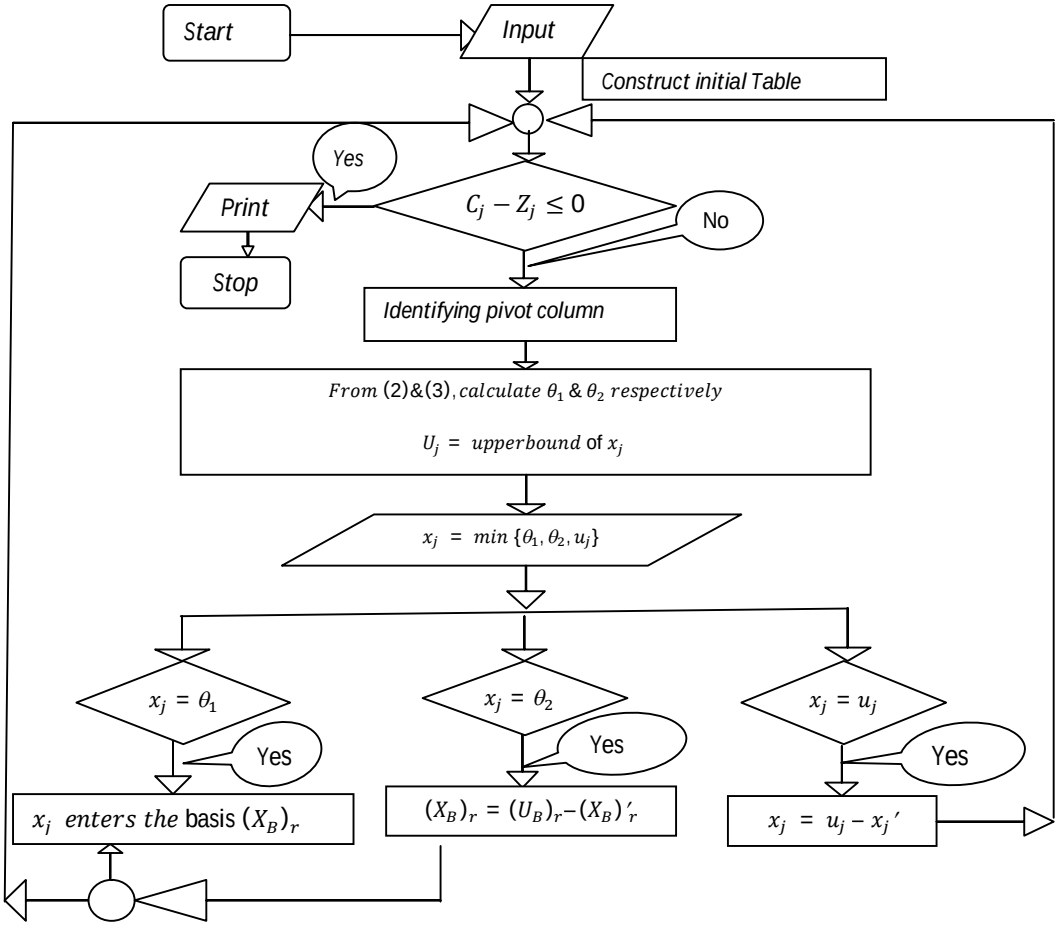
where,

$$0 \leq (X_B)'_r \leq (U_B)'_r$$

Sub-step 3: If $\theta = u_j$, x_j is substituted at its upper bound by $x_j = u_j - x'_j$ while remaining non-basic.

❖ **Flowchart:**

The flowchart of the developed code is presented in this section.



V. Conclusion

Now-a-days computer programming plays an important role for solving various problems in Operations Research and is a much needed tool for solving large scale problems. In this paper we have discussed the algorithm for solving LPP with Bounded Variables, have developed a MATHEMATICA program for solving such problems. It is evident from table 6 that the developed program can reduce significantly the time taken to provide optimum solution. Despite the restrictions, this program may be used more efficiently for solving LPP with bounded variables.

---

1. Dantzig, G. B., 1955, Upper Bounds, Secondary Constraints and Triangularity in Linear Programming, Econometrica, **23**, No. 2.
2. Wagner, H. M., 1958, The Dual Simplex Algorithm for Bounded Variables, Nav. Res. Log. Quart. **5**, 257-261.
3. Maros I, 2003a, A piecewise linear dual phase-I algorithm for the simplex method, Computational Optimization and Applications, **26**, 63-81.
4. Maros I, 2003b, A generalized dual phase-2 simplex algorithm, European Journal of Operational Research, **149**, 1-16.
5. C. Duguay, M. Todd and H.M. Wagner, 1972/73, Linear programming with relative bounded variables, Management Sci., **19**, 751-759.
6. Youshen Xia and Jiasong Wang, 1995, Neural Network for Solving Linear Programming Problems with Bounded Variables, IEEE transaction on neural networks, **6(2).**
7. Roni Saha and M. Babul Hasan, 2010, "A Computer Technique for Sensitivity Analysis in Linear Programs, Dhaka Univ.J.Sci. **58(2)**, 279-286.
8. Hasan, M.B. Hasan, 2008, "Solution Linear Fractional Programming Problems through Computer Algebra, The Dhaka University Journal of Science, **57(1)**, 23-28.
9. Morshed, M.B. Hasan, 2005, Graphical representation of feasible region of Linear programming problems using Mathematica, The Dhaka University Journal of Science, **53(1)**, 87-96.
10. Taha H. A., 1999, Operation Research An Introduction, Prentice-Hall of India Pvt. Ltd, New Delhi.
11. Gupta, P.K. & Hira, D.S., 1998, Problems in Operations Research, S. Chand & Company, New Delhi.