LINEAR PROGRAMMING

Description of some of the function blocks:

Some of the functional block of MATLAB are called in every algorithm , they are input.m, Br.m, vr.m, MRT.m, MRTD.m.

```
function [type,c,A,rel,b]=Input(type,c,A,rel,b)
%This function is made to enter the input parameters which represents the
%standard linear program
%(LP): min c'x
%       subject to Ax=b, x>0
clc;
clear all;
clc;
disp('Enter the input parametrs')
disp('-------------------------')
type = input('Please enter type of the objective function:')
c = input('Enter the cost values,c:')
A = input('Enter the constraint matrix A:')
rel = input('Enter the type of the constraint equations,rel:')
b = input('Enter the right side of the constraint equation, b :')
disp('Input parameters are entered')
disp('----------------------------')
```

```
function [m, j] = Br(d)
% this function finds the first negative number in the tableau to perform
% pivot operation since it is not mandatory to choose the most negative
% number
% Output parameters:
% m - first negative number in the array d
% j - index of the entry m.
%ind = find(d < 0);
x=min(d);
ind=find(d==x);
```

```
if ~isempty(ind)
    j = ind(1);
    m = d(j);
else
    m = [];
    j = [];
end

function e = vr(m,i)
% The ith coordinate vector e in the m-dimensional Euclidean space.
e = zeros(m,1);
e(i) = 1;

function [row, mi] = MRT(b,a)
% The Minimum Ratio Test (MRT) performed on vectors a and b for the simplex method
% Output parameters:
% row – index of the pivot row
% mi – value of the smallest ratio.
 m = length(b); % gives the length of the pivot element a
c = 1:m;       % contains the total number of elements in the column of b
l = c(a > 0);  % takes the element position which has the positive value inorder to find the pivot element
[mi, row] = min(b(l)./a(l));  % finding the pivot element by the equation min { b / pivot column elements}
row = l(row);                 % gives the row containg the pivot element


function col = MRTD(a, b)
% The Maximum Ratio Test performed on vectors a and b.
% This function is called from within the function dualsimplex.
% Output parameter:
% col - index of the pivot column.
m = length(a);
c = 1:m;
l = c(b < 0);
[mi, col] = max(a(l)./b(l));
```

```
col = l(col);
```

1. TWO PHASE METHOD:

The twophase.m performs two-phase simplex operation.

```
function twophase
%This alogorithm solves the linear programming using 2-phase method
% as per the standard linear programming (LP)
% (LP):        min(or max) z = c'x
%                    Subject to Ax=b
%                        x >= 0
%  Since the 2-phase method is used to find the optimal basic feasible
%  solution along with finding the initial basic feasible solution using
%  phase-1.
% Phase-2: used to find the final optimal solution
[type,c,A,rel,b]=Input();
if (type == 'min')  % to check the type of the linear program
    mm = 0;          % variable mm=0 if the type of the problem is minimum
else
    mm = 1;          % variable mm=0 if the type of the problem is maximum
    c = -c;         % the cost function is multiplied with negative to convert it back to the type minimum form
end

b = b(:);          % converting the b into column matrix b=transpose(b)
[m, n] = size(A);   % estimating the number of rows and columns of the matrix
n1 = n;
les = 0;            % variable to count the number of constraints of type <
neq = 0;            % variable to count the number of constraints of type =
red = 0;            % variable to indicate redundant solution

if length(c) < n    % if length of c not equal to number of columns
    c = [c zeros(1,n-length(c))]; %   make the number of elements in the cost function equal to column of A
end
for i=1:m           % m represents the number of constraint equations
    if(rel(i) == '<')    %  checking the constraint type like >= , = or <=
```

```
        A = [A vr(m,i)];     % if constraint type is < then add the slack variable  using function vr.
        les = les + 1;       % Update the variable which has count of constraints of type <
    elseif(rel(i) == '>')
        A = [A -vr(m,i)];    % if constraint type is > then add the surplus variable using function vr,
    else
        neq = neq + 1;       % counts the number of constraint equations of type '='
    end
end
ncol = length(A);    %updating the number of columns
if les == m          % if the number of slack variables is equal to the total number of constraint equations
    c = [c zeros(1,ncol-length(c))];   %   length of c equal to ncol of A after adding slack or surplus variables
    A = [A;c];                  % adding cost  c as the extra row to matrix A to form the canonical
    A = [A [b;0]];              % adding b  to form complete canonical representation
    [subs, A, z] = loop(A, n1+1:ncol, mm, 1, 1);  % function to solve the given linear program

    disp('          End of primal simplex ')
    disp(' **********************************')
else
    A = [A eye(m) b];      %   add the identity matrix for those of constraint type '>' and '='
    w = -sum(A(1:m,1:ncol)); % building the artificial cost function for phase-1 by
    c = [c zeros(1,length(A)-length(c))]; %adding zero cost functions to added identity matrix
    A = [A;c];                       % appending to form canonical form in phase 1
    A = [A;[w zeros(1,m) -sum(b)]];   % appending the calculated artificial cost values w to  canonical
    subs = ncol+1:ncol+m;                %  location number of the basic variables
    av = subs;                       % to compare in the future for the redundancy
    [subs, A, z] = loop(A, subs, mm, 2, 1);  % function to solve the given linear program

    disp('          End of Phase-1')
    disp(' **********************************')

    nc = ncol + m + 1;                  % total number of columns including [A I:b]
    x = zeros(nc-1,1);                 %  initializing the values of x to zero to update them in the phase-2;  Ax=b
    x(subs) = A(1:m,nc);               % transferring the x values from the output of phase-1
    xa = x(av);                       % extracting the initial z value which is zero at the beginning of phase-2
```

```
    com = intersect(subs,av);       % to check whether the non-basic elements are removed after the pivoting
    if (any(xa) ~= 0)               % if the initial z value after phase-1 is not zero
        disp(sprintf('\n\n infeasible solution exist \n'))  % declaring the problem as infeasible linear program
        return
    else
        if ~isempty(com)              % to check whether the non-basic elements variables are zero if not this
results in redundancy
            red = 1;                % non-basic elements are not yet been removed so there is redundancy
        end
    end
    A = A(1:m+1,1:nc);               % removing last row which represented as the artificial cost function in
the phase-1
    A =[A(1:m+1,1:ncol) A(1:m+1,nc)];    % appending original A with the final b value of phase-1
    [subs, A, z] = loop(A, subs, mm, 1, 2);  % pivot operation in phase-2
    disp(' End of Phase-2')
    disp(' *********************************')
end

if (z == inf | z == -inf)              % if the final cost is infinity then stop the process
    return
end
[m, n] = size(A);                    % final size of the tableau at the end of phase-2
x = zeros(n,1);                      % flushing out the old values stored for the variable x
x(subs) = A(1:m-1,n);                % Updating the final basic variable values
x = x(1:n1);                         % obtaining the optimal x values for the actual number of variables
if mm == 0
    z = -A(m,n);                     % if the objective is minimum then obtain positive optimal cost value
else
    z = A(m,n);                      % if the objective is maximum then obtain negative optimal cost value
end
disp(sprintf('\n\n Problem has a finite optimal solution\n'))
disp(sprintf('\n The values of the basic variables are:\n'))
for i=1:n1
    disp(sprintf(' x(%d)= %f ',i,x(i)))
```

```
end
disp(sprintf('\n Optimal Objective value:\n'))
disp(sprintf(' z= %f',z))
t = find(A(m,1:n-1) == 0);     % testing for the total number of solutions
if length(t) > m-1           % if the number of feasible variables is greater than the number of constraint
equation
   str = 'Problem has infinitely many solutions';
   msgbox(str,'Warning Window','warn')
end
if red == 1                  % if the slack or surplus elements are not reduced at the end
   disp(sprintf('\n Constraint system is redundant\n\n'))  % then declare problem is redundant
end
      varargout(1)={subs(:)}; % if the row is positive , gives the location of the basic variables
      varargout(2)={A};              % final A value
      varargout(3) = {x};      % flush all the  basic variables x*
      varargout(4) = {z};            % final cost value z*
```

Loop which performs the phase-1 and Phase-2 operation:

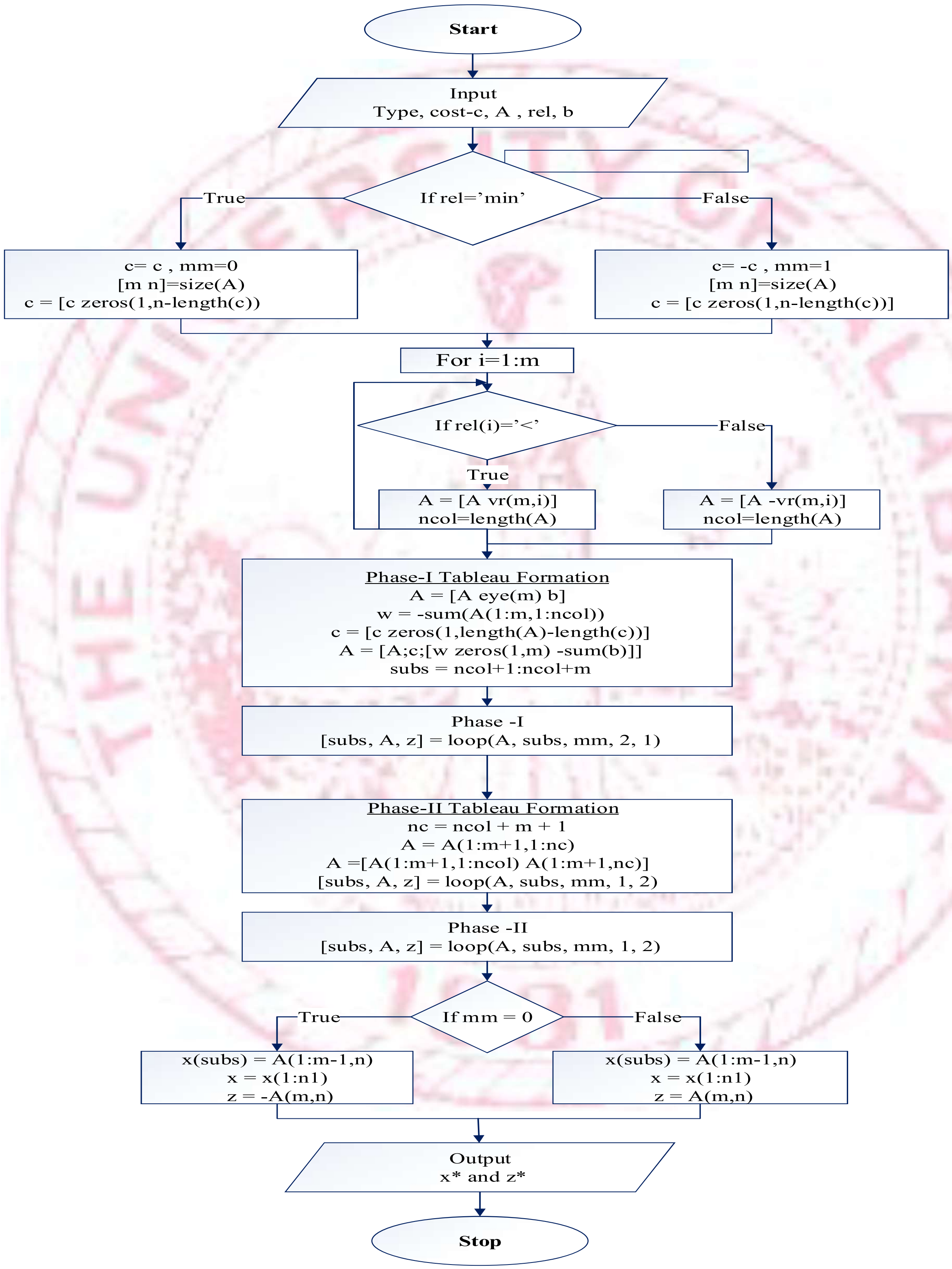
```
function [subs, A, z]= loop(A, subs, mm, k, ph)
% Main loop of the simplex primal algorithm.
tbn = 0;  % initialize the tableu number
if  ph == 1   %if the loop is executing phase-1
   disp(sprintf('\n\n Initial tableau'))
   A
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
end

if ph == 2    % if the loop is executing phase-2
   tbn = 1;
   disp(sprintf('\n\n Tableau %g',tbn))
   A
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
```

```
end
[m, n] = size(A);  % number of rows and columns
[mi, col] = Br(A(m,1:n-1));  %Function to find the most negative pivot column
 while ~isempty(mi) & mi < 0 & abs(mi) > eps % if the most negative number in that column is negative
    t = A(1:m-k,col);                    % extracting the elements corresponding to the pivot column
    if all(t <= 0)                       %if the elements in the pivot column are negative or zero
        if mm == 0                       % if the objective type is minimum
            z = -inf;                    % set objective value equal to negative infinite
        else
            z = inf;                     % if the objective type is maximum then set objective value as positive infinity
        end
        disp(sprintf('\n Unbounded optimal solution with z=%s\n',z)) % print the linear program is unbounded
        return
    end
    [row, small] = MRT(A(1:m-k,n),A(1:m-k,col));  % pivot element by considering the b and the pivot col
    if ~isempty(row)                     % if there is a pivot element
        if abs(small) <= 100*eps & k == 1  %   avoid the cycling and to avoid having unbounded solution
            [s,col] = Br(A(m,1:n-1));          % Function to find the most negative pivot column
        end
        disp(sprintf(' pivot row-> %g pivot column->%g',row,col))
        A(row,:)= A(row,:)/A(row,col); % performing pivot operation for the row associated to pivot element .
        subs(row) = col;                 % add the location of the basic element
        for i = 1:m                      %  Pivot operation
            if i ~= row
               A(i,:)= A(i,:)-A(i,col)*A(row,:); % pivot operation to other rows of the Tableau
            end
        end
        [mi, col] = Br(A(m,1:n-1));    % process of finding pivot column
    end
    tbn = tbn + 1;                       % updating the tableau number
    disp(sprintf('\n\n Tableau %g',tbn))
    A
    disp(sprintf(' Press any key to continue ...\n\n'))
    pause
```

end

z = A(m,n);

Flowchart:



Results:

$$\begin{aligned} \text{minimize} \quad & 4x_1 + x_2 + x_3 \\ \text{subject to} \quad & 2x_1 + x_2 + 2x_3 = 4 \\ & 3x_1 + 3x_2 + x_3 = 3 \\ & x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0. \end{aligned}$$

Enter the input parameters

Please enter type of the objective function: 'min'

type =

min

Enter the cost values, c:[4 1 1]

c =

4 1 1

Enter the constraint matrix A:[2 1 2;3 3 1]

A =

2 1 2

3 3 1

Enter the type of the constraint equations, rel:'=='

rel = ==

Enter the right side of the constraint equation, b :[4 3]

b = 4 3

Input parameters are entered

Initial tableau

A =

2	1	2	1	0	4
3	3	1	0	1	3
4	1	1	0	0	0
-5	-4	-3	0	0	-7

```
Press any key to continue ...

pivot row-> 2 pivot column->1
Tableau 1
A =
        0   -1.0000    1.3333    1.0000   -0.6667    2.0000
   1.0000    1.0000    0.3333         0    0.3333    1.0000
        0   -3.0000   -0.3333         0   -1.3333   -4.0000
        0    1.0000   -1.3333         0    1.6667   -2.0000

Press any key to continue ...

pivot row-> 1 pivot column->3
Tableau 2
A =
        0   -0.7500    1.0000    0.7500   -0.5000    1.5000
   1.0000    1.2500         0   -0.2500    0.5000    0.5000
        0   -3.2500         0    0.2500   -1.5000   -3.5000
        0         0         0    1.0000    1.0000         0

Press any key to continue ...

          End of Phase-1
*********************************

Tableau 1
A =
        0   -0.7500    1.0000    1.5000
   1.0000    1.2500         0    0.5000
        0   -3.2500         0   -3.5000

Press any key to continue ...

pivot row-> 2 pivot column->2
```

Tableau 2

```
A =

    0.6000         0    1.0000    1.8000
    0.8000    1.0000         0    0.4000
    2.6000         0         0   -2.2000
```

Press any key to continue ...

End of Phase-2

****

Problem has a finite optimal solution

The values of the basic variables are:

x(1)= 0.000000

x(2)= 0.400000

x(3)= 1.800000

Optimal Objective value:

z= 2.200000

2. Revised simplex Method:

```
function revsim
%This algorithm solves the linear programming revised simplex method
% as per the standard linear programming (LP)
% (LP):        min(or max) z = c'x
%                   Subject to Ax=b
%                       x >= 0
%
% solution along with finding the initial basic feasible solution using
% phase-1.
%
clc;
clear all;
[type,c,A,rel,b]=Input()

if (type == 'min')  % to check the type of the linear program
```

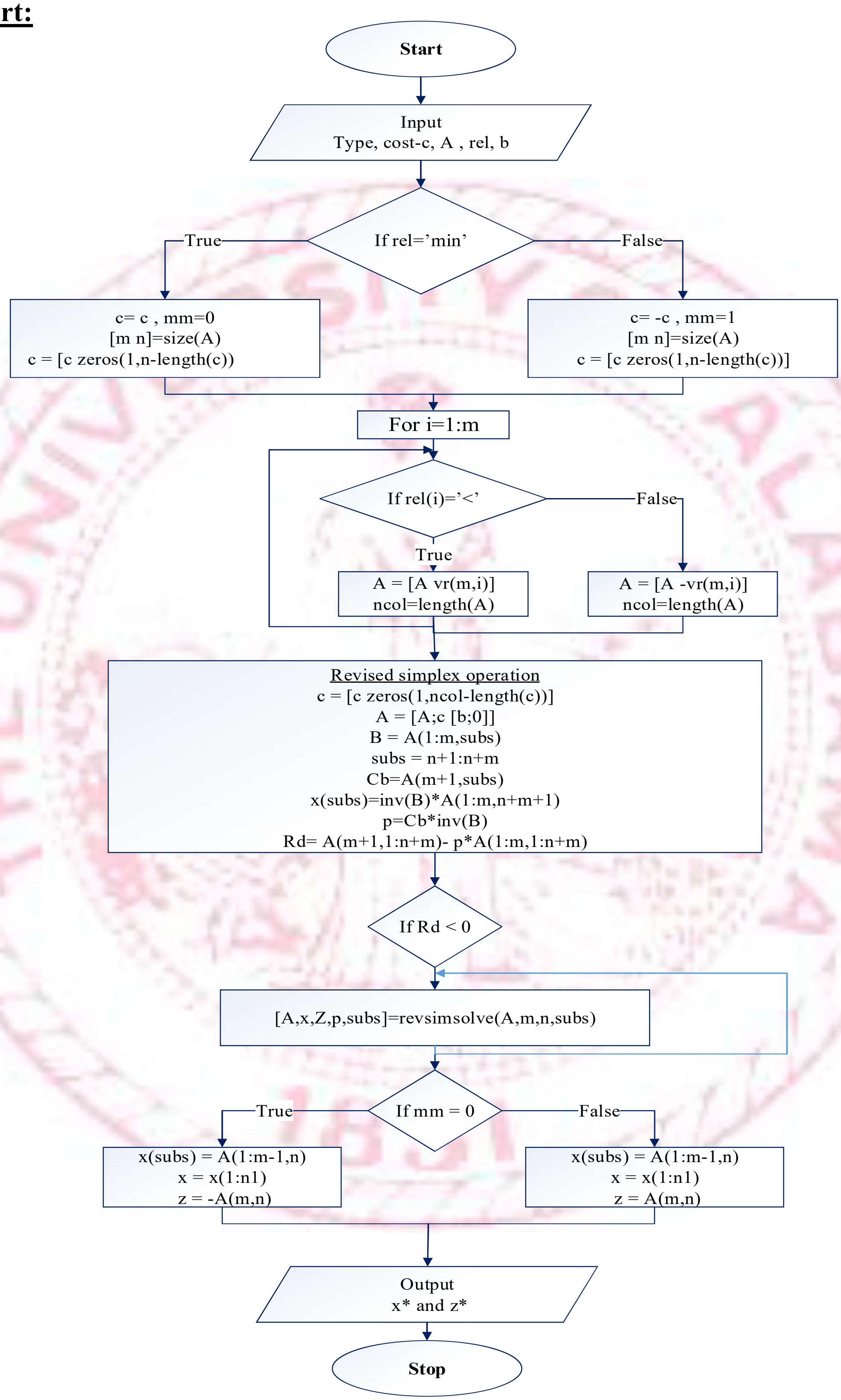
```
    mm = 0;             % variable mm=0 if the type of the problem is minimum
else
    mm = 1;             % variable mm=0 if the type of the problem is maximum
    c = -c;             % the cost function is multiplied with negative to convert it back to the type minimum form
end

b = b(:);           % converting the b into column matrix b=transpose(b)
[m, n] = size(A);   % estimating the number of rows and columns of the matrix
n1 = n;
les = 0;            % variable to count the number of constraints of type <
neq = 0;            % variable to count the number of constraints of type =
red = 0;            % variable to indicate redundant solution

if length(c) < n    % if length of c not equal to number of columns
    c = [c zeros(1,n-length(c))]; %   make the number of elements in the cost function equal to column of A
end


for i=1:m           % m represents the number of constraint equations
    if(rel(i) == '<')    %  checking the constraint type like >= , = or <=
        A = [A vr(m,i)];     % if constraint type is < then add the slack variable  using function vr.
        les = les + 1;       % Update the variable which has count of constraints of type <
    elseif(rel(i) == '>')
        A = [A -vr(m,i)];    % if constraint type is > then add the surplus variable using function vr,
    else
        neq = neq + 1;       % counts the number of constraint equations of type '='
    end
end

ncol = length(A);    %updating the number of columns


if les == m         % if the number of slack variables is equal to the total number of constraint equations
    c = [c zeros(1,ncol-length(c))];   %  length of c equal to ncol of A after adding slack or surplus variables
```

```
    A = [A;c];                    % adding cost equation c as the extra row to matrix A to form the canonical
    A = [A [b;0]];                % adding b also to form complete canonical representation
    subs = n+1:n+m;               % columns of the basic variables
    [A,x,Z,p,subs]=revsimsolve(A,m,n,subs);    % revised simplex method solver
    disp(sprintf(' final tableu B inverse and xb is ---> '));
    A(1:m,n+1:end)
    disp(sprintf(' final dual variables are labmda* ---->  '));
    lambda=p'
    disp(sprintf(' final Basic feasible solution is x* ---->  '));
    x
    disp(sprintf(' final Basic feasible solution is Z*---->  '));
    Z
end

function [A,x,Z,p,subs]=revsimsolve(A,m,n,subs)
% this function is to perfrom the pivot operations for
% solving the revised simplex method
    B=A(1:m,subs);
    invB=inv(B);
    Cb=A(m+1,subs);               % basic variable cost
    x(subs)=inv(B)*A(1:m,n+m+1);    % basic varible vector
    p=Cb*inv(B);                  % dual vector
    Rd= A(m+1,1:n+m)- p*A(1:m,1:n+m) % reduced cost
    [j col]= Br(Rd);              % location and value of most negative number

while ~isempty(j) & j < 0 & abs(j) > eps                    % if there is any negative cost

Jin=Rd(col);
    % Compute the vector u (i.e., the reverse of the basic directions)
    u = inv(B)*A(1:m,col);        % pivot column extraction
    I = find(u > 0);
    if (isempty(I))
        f_opt = -inf;  % Optimal objective function cost = -inf
        x_opt = [];       % Produce empty vector []
```

```
        return       % Break from the function
    end
    [row, min] = MRT(A(:,n+m+1),u);  % find the minimum ratio
    x(subs)=x(subs)'-min*u; % pivot operation
    x(col)= min;        % add the new value of the basic varioable
    subs(row)=col        % update the location of the basic variable
    disp(sprintf(' pivot row-> %g pivot column->%g',row,col));
    disp(sprintf('initial B inverse and Xb tablue ----> '));
    A(1:m,n+1:end)
    A(row,:)= A(row,:)/A(row,col);% performing pivot operation for the row associated to pivot element .
    subs(row) = col;             % adding the location of new basic variable
    for i = 1:m                  %  pivot operation for the rest of the rows of the tableau
      if i ~= row
        A(i,:)= A(i,:)-A(i,col)*A(row,:); % Pivot operation
      end
    end
    invB=A(1:m,n+1:n+m)
    Cb=A(m+1,subs);
    x(subs)= A(1:m,n+m+1);
    p=Cb*invB ;
    Rd= A(m+1,1:n+m)- Cb*A(1:m,1:n+m);   % reduced cost
    [j col]= Br(Rd);             % checking for the negative reduced cost
end
x=x(1:n)';
Z=-Cb*A(1:m,n+m+1);
```

LINEAR PROGRAMMING

Flowchart:



Results:

Maximize $3x_1 + x_2 + 3x_3$ subject to

$$\begin{aligned} 2x_1 + x_2 + x_3 &\leqslant 2 \\ x_1 + 2x_2 + 3x_3 &\leqslant 5 \\ 2x_1 + 2x_2 + x_3 &\leqslant 6 \\ x_1 \geqslant 0, \quad x_2 \geqslant 0, \; x_3 &\geqslant 0. \end{aligned}$$

Enter the input parametrs

Please enter type of the objective function:'max'

type =

max

Enter the cost values,c:[3 1 3]

c =

3 1 3

Enter the constraint matrix A:[2 1 1;1 2 3;2 2 1]

A =

2 1 1

1 2 3

2 2 1

Enter the type of the constraint equations,rel:'<<<'

rel =

<<<

Enter the right side of the constraint equation, b :[2 5 6]

b = 2 5 6

Input parameters are entered

Rd = -3 -1 -3 0 0 0

subs = 1 5 6

pivot row-> 1 pivot column->1

initial B inverse and Xb tablue ---->

1 0 0 2

```
     0     1     0     5
     0     0     1     6
invB =    0.5000         0         0
         -0.5000    1.0000         0
         -1.0000         0    1.0000

subs =    1     3     6
 pivot row-> 2 pivot column->3
initial B inverse and Xb tablue ---->
    0.5000         0         0    1.0000
   -0.5000    1.0000         0    4.0000
   -1.0000         0    1.0000    4.0000

invB =
    0.6000   -0.2000         0
   -0.2000    0.4000         0
   -1.0000         0    1.0000
 final tableu B inverse and xb is ---->
    0.6000   -0.2000         0    0.2000
   -0.2000    0.4000         0    1.6000
   -1.0000         0    1.0000    4.0000
 final dual variables are labmda* ---->
lambda =   -1.2000
           -0.6000
                 0
 final Basic feasible solution is x* ---->
x =    0.2000
            0
       1.6000
 final Basic feasible solution is Z*---->
Z =   5.4000
```

3. Dual simplex Method:

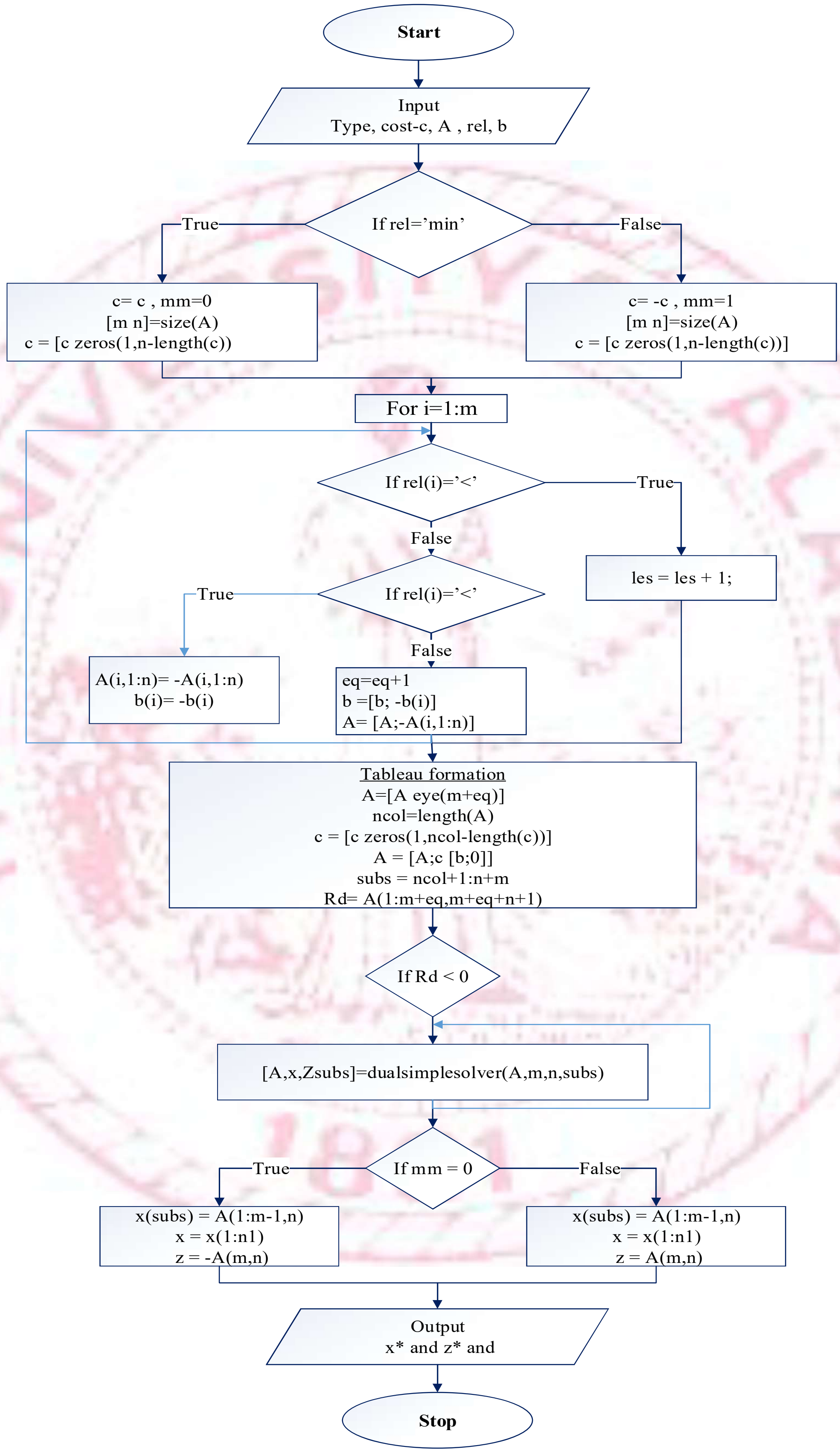
```
function dualsimplex
% this function perform the dualsimplex method for a linear program
% the standard linear program of ----> type  c'x
%                                   subject to Ax rel b
%                                    x>0
% where type represent the type of the cost function min or max
% where rel could be <,> or = sign in the cosntraint equation
% c is the cost column matrix
% x is the basic variable column matrix

clc;
[type,c,A,rel,b]=Input()   % function to input the initial parameters

if (type == 'min')  % to check the type of the linear program
   mm = 0;           % variable mm=0 if the type of the problem is minimum
else
   mm = 1;           % variable mm=0 if the type of the problem is maximum
   c = -c;           % the cost function is multiplied with negative to convert it back to the type minimum form
end

b = b(:);          % converting the b into column matrix b=transpose(b)
[m, n] = size(A);   % estimating the number of rows and columns of the matrix A
n1 = n;
les = 0;            % variable to count the number of constraints of type <
neq = 0;            % variable to count the number of constraints of type =
red = 0;            % variable to indicate redundant solution
eq = 0;
if length(c) < n    %  length c less than the column matrix A
   c = [c zeros(1,n-length(c))]; %  make size of column equal to column of matrix A
end

for i=1:m           % m represents the number of constraint equations
```

```
    if(rel(i) == '<')    % for each constraint equation checking the constraint type like >= , = or <=
        les = les + 1;      % Update the variable which has count of constraints of type <
    elseif(rel(i) == '>')
        A(i,1:n)= -A(i,1:n);
        b(i)= -b(i);        % changing the sign of the RHS of Ax>=b
        neq=neq+1;  % counts the number of constraint equations of type '='
    else
        eq=eq+1;        % number of equality constraints
        b =[b; -b(i)];  % add one -b(i) to b vector
        A= [A;-A(i,1:n)];  % update the A after adding a row correspond to the equality constraint
    end
end
A=[A eye(m+eq)];         % adding the identity matrix for whole A
ncol = length(A);    % length of Matrix A
c = [c zeros(1,ncol-length(c))];   % make the length of c equal to ncol of A
A = [A ;c];                   % adding cost equation c  to  canonical representation
A = [A [b;0]];               % adding b also to form complete canonical representation of the given linear program
subs = ncol+1:ncol+m;              %  this indicates column number which has identity matrix possess as basic
variables
disp(sprintf('\n\n Initial tableau'))
A
disp(sprintf(' Press any key to continue ...\n\n'))
pause
[bmin, row] = Br(b);               % to find the min b of RHS in Ax>=b
tbn = 0;                            % initialize the number of tableau

Dual-simplex operation% Pivot operation
while ~isempty(bmin) & bmin < 0 & abs(bmin) > eps
    if A(row,1:m+eq+n) >= 0           % checking whether the row particular to negative b is not positive
        disp(sprintf('\n\n Empty feasible region\n'))
        varargout(1)={subs(:)};        % if the row is positive , position of the basic variables
        varargout(2)={A};              % final A value
        varargout(3) = {zeros(n,1)};    % flush all the  basic variables to zeros
        varargout(4) = {0};             % final cost value set as zero
```

```
        return
    end
    col = MRTD(A(m+1+eq,1:m+eq+n),A(row,1:m+eq+n));     %to find the pivot element
    disp(sprintf(' pivot row-> %g pivot column-> %g',row,col))
    subs(row) = col;                          % the pivot element gets added to basic
      A(row,:)= A(row,:)/A(row,col);  % making pivot element equal to 1
    for i = 1:m+eq+1                    %  Update the tableau
        if i ~= row
            A(i,:)= A(i,:)-A(i,col)*A(row,:);  % pivot operation
        end
    end
    tbn = tbn + 1;              %update the number of tableau
    disp(sprintf('\n\n Tableau %g',tbn))
    A
    disp(sprintf(' Press any key to continue ...\n\n'))
    pause
    [bmin, row] = Br(A(1:m+eq,m+eq+n+1));  %     i.e process of finding pivot column
end

x = zeros(m+n+eq,1);                % flushing out the old values stored for the variable x
x(subs) = A(1:m+eq,m+eq+n+1);            %% Updating the final basic variable values
x = x(1:n);
if mm == 0
    z = -A(m+eq+1,n+m+eq+1);    % if the objective is minimum
else
    z = A(m+eq+1,n+m+eq+1);     % if the objective is maximum
end
disp(sprintf('\n\n Problem has a finite optimal solution\n\n'))
disp(sprintf('\n Values of the legitimate variables:\n'))
for i=1:n
    disp(sprintf(' x(%d)= %f ',i,x(i)))     %print the basic variables
end
disp(sprintf('\n Objective value at the optimal point:\n'))
disp(sprintf(' z= %f',z))
```

LINEAR PROGRAMMING

Flowchart:



Results:

$$
\begin{aligned}
\text{minimize} \quad & 3x_1 + 4x_2 + 5x_3 \\
\text{subject to} \quad & x_i + 2x_2 + 3x_3 \geqslant 5 \\
& 2x_1 + 2x_2 + \ \ x_3 \geqslant 6 \\
& x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0
\end{aligned}
$$

Enter the input parametrs

Please enter type of the objective function:'min'

type =min

Enter the cost values,c:[3 4 5]

c = 3 4 5

Enter the constraint matrix A:[1 2 3;2 2 1]

A =

1 2 3

2 2 1

Enter the type of the constraint equations,rel:'>>'

rel = >>

Enter the right side of the constraint equation, b :[5 6]

b = 5 6

Input parameters are entered

Initial tableau

A =

-1	-2	-3	1	0	-5
-2	-2	-1	0	1	-6
3	4	5	0	0	0

Press any key to continue ...

pivot row-> 2 pivot column-> 1

Tableau 1

A =

0	-1.0000	-2.5000	1.0000	-0.5000	-2.0000
1.0000	1.0000	0.5000	0	-0.5000	3.0000
0	1.0000	3.5000	0	1.5000	-9.0000

```
Press any key to continue ...
pivot row-> 1 pivot column-> 2

Tableau 2
A =
    0    1.0000    2.5000   -1.0000    0.5000    2.0000
    1.0000         0   -2.0000    1.0000   -1.0000    1.0000
         0         0    1.0000    1.0000    1.0000  -11.0000
Press any key to continue ...
Problem has a finite optimal solution

Values of the legitimate variables:
x(1)= 1.000000
x(2)= 2.000000
x(3)= 0.000000

Objective value at the optimal point:
z= 11.000000
```

4. Primal-Dual Algorithm:

This is the most powerfull algorithm to solve linear programming. The input to algorithms is given by calling the function InputPrimar.m . where as primaldual.m handles the initial tableu formation and generating the final output values and primaldualsolver.m solves and updates the tableau.

```
function [type,c,A,rel,b,ld]=Input(type,c,A,rel,b,ld) ;
%This function is made to enter the input parameters which represents the
%standard linear program
%(LP): min c'x
%       subject to Ax=b, x>0
clc;
clear all;
clc;
disp('Enter the input parametrs')
```

```
disp('-------------------------')
type = input('Please enter type of the objective function:')
c = input('Enter the cost values,c:')
A = input('Enter the constraint matrix A:')
rel = input('Enter the type of the constraint equations,rel:')
b = input('Enter the right side of the constraint equation, b :')
ld = input('Enter the initial labmda, Lambda :')

disp('Input parameters are entered')
disp('----------------------------')

function [A,P,Pc]=primaldualsolver(A,P,Pc,m,n,)
% performs the operation of updating the tableu by performing the pivot
% operation for the primal-dual operaton
%Input:
% A: Initial Tableau
% P: Position of the basic variables
% Pc: Position of the non-basic variables
%Output:
% X: final basic solution
% P: location of the final basic variables
% Z*: final optimal cost
ncol=length(A);
tbn = 0;                          % intialize the number of tableu

while any(A(m+1,n+1:ncol-1)== 0) | any(A(m+1,1:n)< 0)
    epsi=(A(m+2,1:n)./(-A(m+1,1:n )));
    [bmin, col] = Br(epsi) ;              % to find the min b of RHS in Ax>=b
    A(m+2,1:n) = A(m+2,1:n) + bmin*(A(m+1,1:n ))  %update ct - lamda*A= ct-lamda*A + epsi(r(d))
    row = MRT(A(1:m+1,ncol),A(1:m+1,col));           %to find the pivot elemet
    disp(sprintf(' pivot row-> %g pivot column-> %g and pivot element is---> %d',row,col,A(row,col)))
    P(row) = col;                   % the pivot element gets added to basic
    Pc= setdiff((1:ncol-1),P);         %
      A(row,:)= A(row,:)/A(row,col);  % making pivot element equal to 1
```

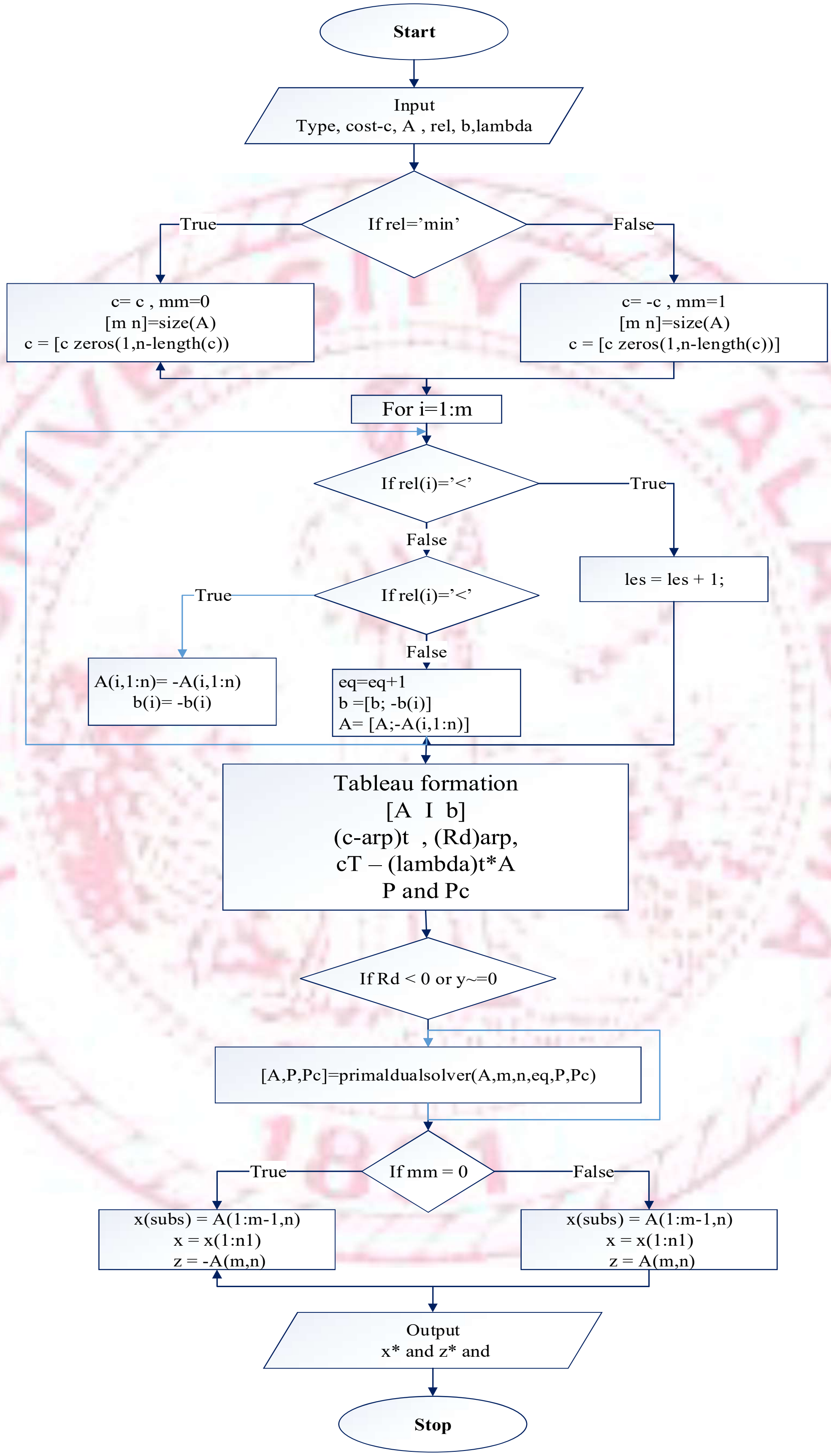
```
    for i = 1:m+1                  %this is to build new canonical form after the pivot operation to the rest of the
elements corresponding to the other rows
        if i ~= row
            A(i,:)= A(i,:)-A(i,col)*A(row,:);  % making the rest of the elements in the pivot column equal to zero
except pivot element
        end
    end
    tbn = tbn + 1;               %update the number of tableu
    disp(sprintf('\n\n Tableau %g',tbn))
    A
    disp(sprintf(' Press any key to continue ...\n\n'))
    pause
end

function primaldual
% this function perform the primaldual  method for a linear program
% the standard linear program of ---->  type  c'x
%                                  subject to Ax rel b
%                                   x>0
% where type represent the type of the cost function min or max
% where rel could be <,> or = sign in the constraint equation
% c is the cost column matrix
% x is the basic variable column matrix

clc;
clear all;
 [type,c,A,rel,b,ld]=InputPrimar();   % function to input the initial parameters

if (type == 'min')  % to check the type of the linear program
    mm = 0;            % variable mm=0 if the type of the problem is minimum
else
    mm = 1;            % variable mm=0 if the type of the problem is maximum
    c = -c;            % the cost function is multiplied with negative to convert it back to the type minimum form
end
```

```
b = b(:);          % converting the b into column matrix b=transpose(b)
[m, n] = size(A);   % estimating the number of rows and columns of the matrix A which helps in building the simplex algorithm
n1 = n;
les = 0;            % variable to count the number of constraints of type <
neq = 0;            % variable to count the number of constraints of type =
red = 0;            % variable to indicate redundant solution
eq = 0;
if length(c) < n    % if number of elements in the cost function is not equal to the number of columns in the matrix A
    c = [c zeros(1,n-length(c))]; % add the zero elements to make the number of elements in the cost function equal to column of A i.e. n
end


for i=1:m           % m represents the number of constraint equations
    if(rel(i) == '<')    % for each constraint equation checking the constraint type like >= , = or <=
        les = les + 1;       % Update the variable which has count of contraints of type <
    elseif(rel(i) == '>')
        A(i,1:n)= -A(i,1:n);
        b(i)= -b(i);         % changing the sign of the RHS of Ax>=b
        neq=neq+1;  % counts the number of contraint equations of type '='
    else
        eq=eq+1;        % number of equality constraints
     end
end
A=[A eye(m)];          % adding the identity marix for whole A
ncol = length(A);    %updating the number of columns after adding or without adding the identity matrix or slack or surplus variables
c = [c zeros(1,ncol-length(c))];   % again adding zero cost elemements to make the lenth of c equal to ncol of A after adding slack or surplus varibles
cld= c-ld*A;               % calculate ct - Lambda*A
A = [A b];             % adding b also to form complete canonical representation of the given linear program
```

```
ncol=length(A)
cin=[zeros(1,n) ones(1,m) zeros(1,1)];
w = -sum(A(1:m,1:ncol))+ cin;  % (r)ARP
cld = [cld zeros(1,ncol-length(cld))];
A=[A;w;cld]; % initial tableu
P = n+1:n+m;                % this indicates column number which has identity matrix posses as basic
variables
Pc= setdiff((1:ncol-1),P);     % column number which has non basic elements
disp(sprintf('\n\n Initial tableau'))
A
disp(sprintf(' Press any key to continue ...\n\n'))
pause

tbn = 0;                    % intialize the number of tableu
[A,P,Pc]=primaldualsolver(A,P,Pc,m,n);   % calling function to solve and update the tableau
x = zeros(m+n,1);                % flushing out the old values stored for the variable x
x(P) = A(1:m,m+n+1);             %% Updating the final basic variable values
x = x(1:n);
c=c(1:n);
if mm == 0
    z = -(c*x);                  % if the objective is minimum then obtain positive optimal cost value
else
    z = (c*x);                   % if the objective is maximum then obtain negative optimal cost value
end

disp(sprintf('\n\n Problem has a finite optimal solution\n\n'))
disp(sprintf('\n Values of the legitimate variables:\n'))
for i=1:n
    disp(sprintf(' x(%d)= %f ',i,x(i)))    %print the basic variables
end
disp(sprintf('\n Objective value at the optimal point:\n'))
disp(sprintf(' z= %f',z))
```

LINEAR PROGRAMMING

Flowchart:



Results:

$$\begin{aligned} \text{minimize} \quad & 2x_1 + x_2 + 4x_3 \\ \text{subject to} \quad & x_1 + x_2 + 2x_3 = 3 \\ & 2x_1 + x_2 + 3x_3 = 5 \\ & x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0. \end{aligned} \quad \boldsymbol{\lambda} = (0, 0)$$

Enter the input parametrs

Please enter type of the objective function:'min'

type = min

Enter the cost values,c:[2 1 4]

c = 2 1 4

Enter the constraint matrix A:[1 1 2;2 1 3]

A =

```
     1     1     2
     2     1     3
```

Enter the type of the constraint equations,rel:'=='

rel = ==

Enter the right side of the constraint equation, b :[3 5]

b = 3 5

Enter the initial labmda, Lambda :[0 0]

ld = 0 0

Input parameters are entered

Initial tableau

A =

```
     1     1     2     1     0     3
     2     1     3     0     1     5
    -3    -2    -5     0     0    -8
     2     1     4     0     0     0
```

Press any key to continue ...

pivot row-> 1 pivot column-> 2 and pivot element is---> 1

Tableau 1

A =

1.0000	1.0000	2.0000	1.0000	0	3.0000
1.0000	0	1.0000	-1.0000	1.0000	2.0000
-1.0000	0	-1.0000	2.0000	0	-2.0000
0.5000	0	1.5000	0	0	0

Press any key to continue ...

A =

1	1	2	1	0	3
1	0	1	-1	1	2
-1	0	-1	2	0	-2
0	0	1	0	0	0

pivot row-> 2 pivot column-> 1 and pivot element is---> 1

Tableau 2

A =

0	1	1	2	-1	1
1	0	1	-1	1	2
0	0	0	1	1	0
0	0	1	0	0	0

Press any key to continue ...

Problem has a finite optimal solution

Values of the legitimate variables:

x(1)= 2.000000

x(2)= 1.000000

x(3)= 0.000000

Objective value at the optimal point:

z= 5.000000

Transportation Problem:

This has some of the common function blocks which is been called commonly. Like NWCR.m which calculates the initial basic solution, dualvariable.m calculates the dual variables, nonbasic.m calculates the reduced cost and cycle.m updates the Tableu .

```
function [x,d,c]= NWCR(a,b,c,s)
% performs the northwest corner rule algorithm
% x ----> shipments using n-w corner rule
% d ----> 1 for basic variable and 0 for non-basic
% a ----> supply
% b ----> demand
% c ----> cost matrix

% check for the balanced transportation problem
[m n]=size(c);
s=s(:);
if (sum(a)~=sum(b))
   disp('Error: unbalanced transportation problem')
   if(sum(a)>sum(b))
      disp('supply is greater than demand, So,add a dummy destination')
      fprintf('dummy destination added b(%d)= %d\n ', n+1,sum(a)-sum(b))
      b = [b sum(a)-sum(b)];
      c =[c s];
      disp('Balanced network is')
      disp(sum(b))
   else
      disp('supply is smaller than demand, So,add a dummy source')
      fprintf('dummy source added b(%d)= %d \n', m+1,sum(b)-sum(a))
      a = [a sum(b)-sum(a)];
      c = [c;s];
      disp('balanced network is')
      disp(sum(a))
   end
```

```
else
    disp('Balanced transportation problem')
    disp(sum(a))
end
m=length(a);
n=length(b);
i=1;
j=1;
x=zeros(m,n);
d=zeros(m,n);
while ((i<=m) & (j<=n))
    if (a(i)< b(j))
        x(i,j)=a(i);
        d(i,j)=1;
        b(j)=b(j)-a(i);
        i=i+1;
    else
        x(i,j)=b(j);
        d(i,j)=1;
        a(i)=a(i)-b(j);
        j=j+1;
    end
end


function [u,v,i,j]=dualvariable(x,d,c)
%[u v] ----> dual variable corresponding to supply and demand
% x ----> the current solution from N_W corner rule
% d ----> indices correspond to basic varibles
% c ----> initial cost matrix
% u ----> correspond to supply
% v ----> correspond to demand

[m n]=size(x);
```

```
if(sum(sum(d))~= m+n-1)
   disp('Error in N-W corner output')
   return
else
   u=Inf*ones(m,1); % assigning an arbitrary values for the u and v
   v=Inf*ones(n,1);  % for the future
   v(n)=0;          % assuming first element of u=0 possessses redundancy
   k=1;
   while k<m+n        % checking the condition with number of basic variables
      for i= m:-1:1
         for j=n:-1:1
            if(d(i,j)>0)
               if(u(i)~=Inf & v(j)==Inf)
                  v(j)=c(i,j)-u(i);
                  k=k+1;
               elseif(u(i)==Inf & v(j)~= Inf)
                  u(i)=c(i,j)-v(j);
                  k=k+1;
               end
            end
         end
      end
   end
end


function [row,col,Rd]=nonbasic(u,v,c,d)
% this program is to find the location of the most negative non-basic
% element Rd
% INPUTS:
% u -----> dual variable of supply
% v -----> dual varible of demand
% c -----> cost matrix
% d -----> indicies of the basic variable
```

```
% OUTPUTS:
% row ---> row of the entering variable
% column ---> colummn of the entering varioable
[m,n]=size(d);
Rd=zeros(m,n);

for i=m:-1:1
    for j=n:-1:1
        if (d(i,j)~=1)
            Rd(i,j)=c(i,j)-(u(i)+v(j));
        end
    end
end
%x=min(Rd<0);
[row col]=find(Rd<0);
if ~isempty(col)
    row=row(1);
    col=col(1);
else
    row = [];
    col = [];
end


function [y,bout]=cycle(x,row,col,b,c,Rd)
% [y,bout]=cycle(x,row,col)
% x: current solution (m*n)
% b: entering basic variables (m*n)
% row,col: index for element entering basis
% y: solution after cycle of change (m*n)
% bout: new basic variables after cycle of change (m*n)
y=x;
bout=b;
while (~isempty(row) & ~isempty(col))
```

```
[m,n]=size(x);
loop=[row col]; % describes the cycle of change
disp('Element entering the basic')
fprintf('R(%d,%d)= %d\n\n',loop(1,1),loop(1,2),Rd(loop(1,1),loop(1,2)))
x(row,col)=Inf; % do not include (row,col) in the search
b(row,col)=Inf;
rowsearch=1; % start searching in the same row
while (loop(1,1)~=row | loop(1,2)~=col | length(loop)==2),
    if rowsearch, % search in row
        j=1;
        while rowsearch
            if (b(loop(1,1),j)~=0) & (j~=loop(1,2))      % element is part of basic solution and not in the same
column
                loop=[loop(1,1) j ;loop]; % add indices of found element to loop
                rowsearch=0; % start searching in columns
            elseif j==n, % no interesting element in this row
                b(loop(1,1),loop(1,2))=0;
                loop=loop(2:length(loop),:); % backtrack
                rowsearch=0;
            else
                j=j+1;                    % update the column
            end
        end
    else % column search  to find the elements which get affected by addition of new basic element
        i=1;
        while ~rowsearch
            if (b(i,loop(1,2))~=0) & (i~=loop(1,1)) % if the element is part of basic solution and not in the
same row
                loop=[i loop(1,2) ; loop];        % add the found element to loop to estimate theta
                rowsearch=1;                    % start searching in rows
            elseif i==m
                b(loop(1,1),loop(1,2))=0;
                loop=loop(2:length(loop),:);        % backward
                rowsearch=1;
```

```
            else
                i=i+1;                  % update the row
            end
        end
    end
end
disp('Step-3')
% compute maximal loop shipment
l=length(loop);
theta=Inf;
minindex=Inf;
for i=2:2:l
    if x(loop(i,1),loop(i,2))<theta,
        theta=x(loop(i,1),loop(i,2));   % calculating the minimum theta value
        minindex=i;
    end;
end
% compute new transport matrix
y(row,col)=theta;
disp('Update Tableu')
for i=2:l-1
    y(loop(i,1),loop(i,2))=y(loop(i,1),loop(i,2))+(-1)^(i-1)*theta;
end
disp('x(updated) =')
disp(y)
bout(row,col)=1;
bout(loop(minindex,1),loop(minindex,2))=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       COMPUTATION OF DUAL VARIABLE    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Step-2(a)')
[u,v]=dualvariable(y,bout,c);
disp('dual of suplly u =')
disp(u')
```

```
    disp('dual of demand v =')
    disp(v')

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %   LOCATION OF BASIC ENTERING ELEMENT    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp('Step-2(b)')
    [row,col,Rd]=nonbasic(u,v,c,bout);
    disp('Reduced cost, Rd =')
    disp(Rd)
end
```

5. <u>Balanced transportation problem:</u>

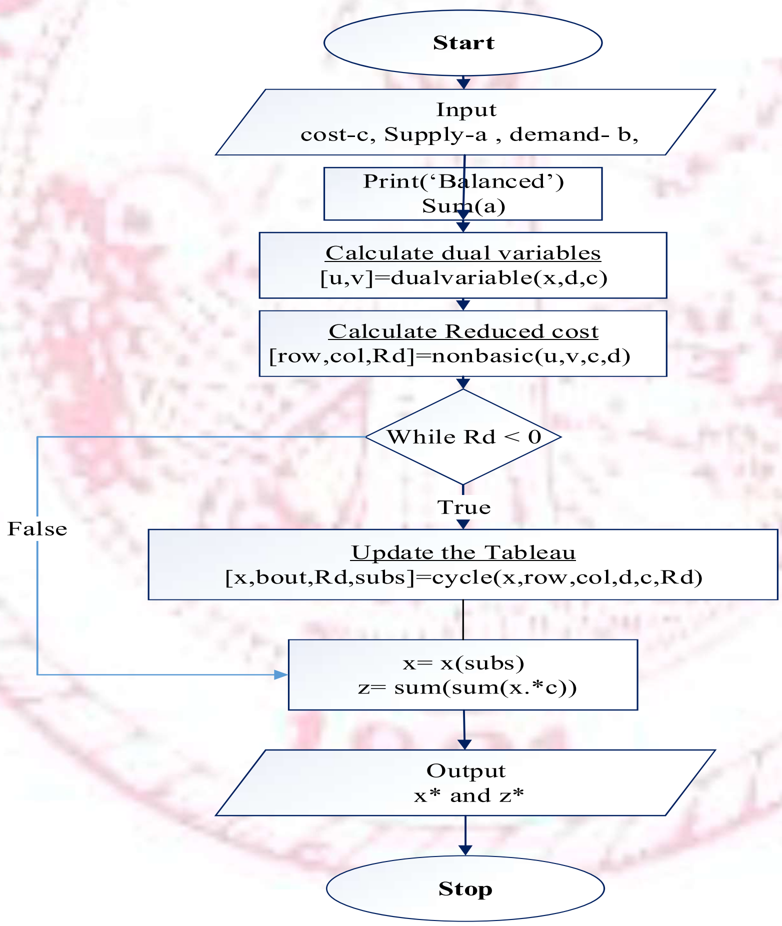
<u>Algorithm:</u>

```
function balancedtransport
% input parameters
% a ---> supply  (m*1)
% b ---> demand  (n*1)
% c ---> cost    (m*n)
% output parameters
% x* ---> optimal solution (m*n)
% c* ---> minimum transportation cost
% s ----> storage cost equal to zero
clc;
clear all;
disp('Enter the input parametrs')
disp('-------------------------')
a = input('Enter the supply, a :')
b = input('Enter the demand, b :')
c = input('Enter the cost values, c:')
s=0;
disp('Input parameters are entered')
disp('----------------------------')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%          NORTH WEST CORNER RULE       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[x,d,c]=NWCR(a,b,c,s);
disp('Initial basic solution ')
disp('x(initial) =')
disp(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        COMPUTATION OF DUAL VARIABLE    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Step-2(a)')
[u,v]=dualvariable(x,d,c);
disp('dual of suplly u =')
disp(u')
disp('dual of demand v =')
disp(v')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   LOCATION OF BASIC ENTERING ELEMENT    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Step-2(b)')
[row,col,Rd]=nonbasic(u,v,c,d);
disp('non basic matrix Rd =')
disp(Rd)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          UPDATING THE TABLEU       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[x,bout]=cycle(x,row,col,d,c,Rd);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          FINAL OUTPUT       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp(' the optimal solution is')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('x* =')
[m n]=size(x);
```

```
for i=1:m
   for j=1:n
      if(bout(i,j)~=0)
         disp(sprintf('x(%d%d)= %d ',i,j,x(i,j)))
      end
   end
end
 disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('Minimum transportation cost is')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
sprintf('Z*= %d\n',sum(sum(x.*c)))
```

Flowchart:



Results:

$$\mathbf{a} = (30,\ 80,\ 10,\ 60)$$

$$\mathbf{b} = (10,\ 50,\ 20,\ 80,\ 20)$$

$$\mathbf{C} = \begin{bmatrix} 3 & 4 & 6 & 8 & 9 \\ 2 & 2 & 4 & 5 & 5 \\ 2 & 2 & 2 & 3 & 2 \\ 3 & 3 & 2 & 4 & 2 \end{bmatrix}$$

```
Enter the input parameters
-------------------------
Enter the supply, a :[30 80 10 60]
a =    30    80    10    60
Enter the demand, b :[10 50 20 80 20]
b =    10    50    20    80    20
Enter the cost values, c:[3 4 6 8 9;2 2 4 5 5;2 2 2 3 2;3 3 2 4 2]
c =
     3     4     6     8     9
     2     2     4     5     5
     2     2     2     3     2
     3     3     2     4     2
Input parameters are entered
----------------------------
Balanced transportation problem
   180
Initial basic solution
x(initial) =
    10    20     0     0     0
     0    30    20    30     0
     0     0     0    10     0
     0     0     0    40    20

Step-2(a)
dual of supply u =   5     3     1     2
dual of demand v =   -2    -1     1     2     0
```

Step-2(b)

non basic matrix Rd =

0	0	0	1	4
1	0	0	0	2
3	2	0	0	1
3	2	-1	0	0

Element entering the basic

R(4,3)= -1

Step-3

Update Tableu

x(updated) =

10	20	0	0	0
0	30	0	50	0
0	0	0	10	0
0	0	20	20	20

Step-2(a)

dual of suplly u = 5 3 1 2

dual of demand v = -2 -1 0 2 0

Step-2(b)

Reduced cost, Rd =

0	0	1	1	4
1	0	1	0	2
3	2	1	0	1
3	2	0	0	0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

the optimal solution is

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x* =

x(11)= 10

x(12)= 20

x(22)= 30

x(24)= 50

x(34)= 10

x(43)= 20

x(44)= 20

x(45)= 20

%%

Minimum transportation cost is

%%

Z*= 610

6. Unbalanced Transportation Problem:

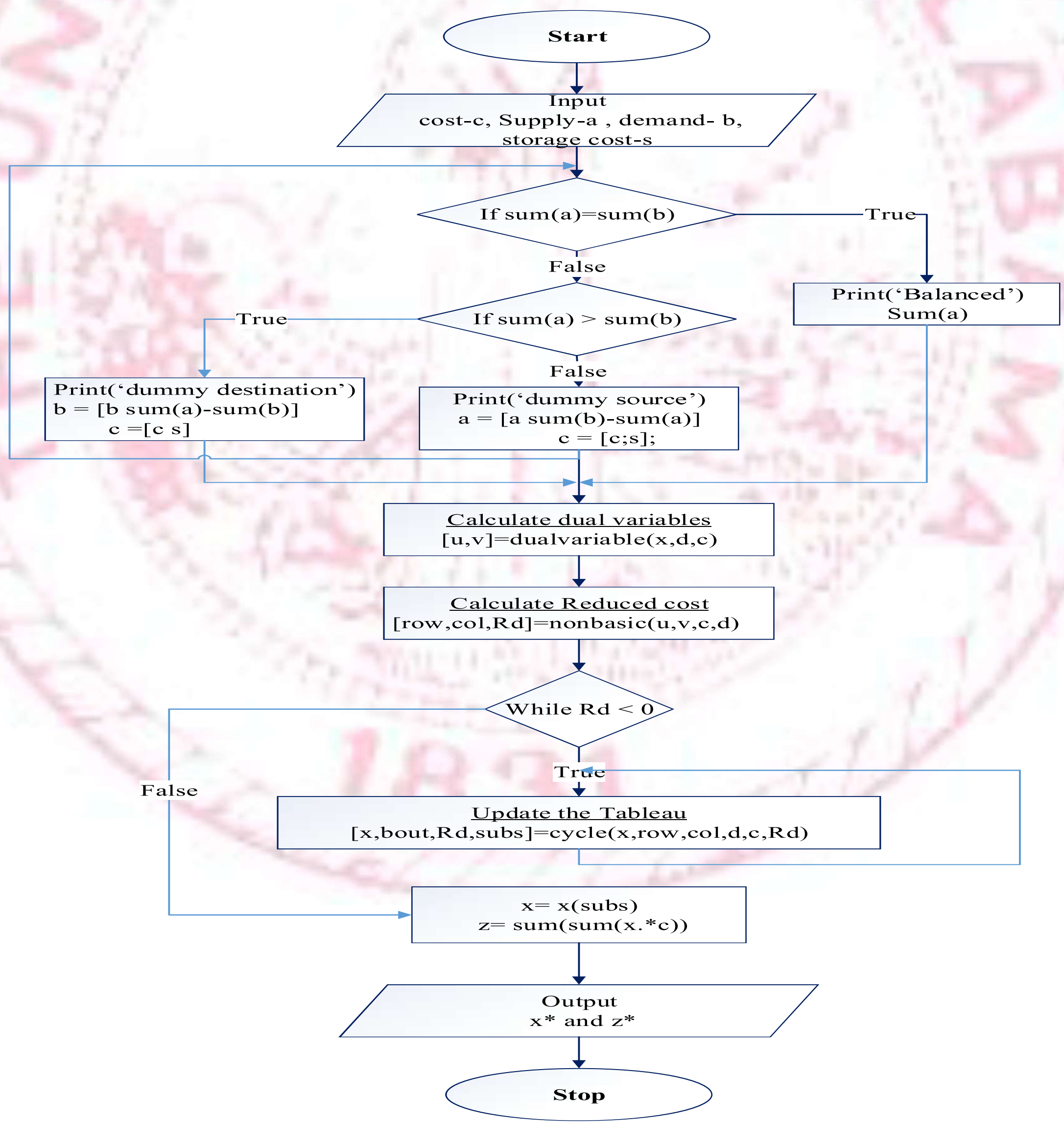
Algorithm:

```
function unbalancetransport
% input parameters
% a ---> supply  (m*1)
% b ---> demand  (n*1)
% c ---> cost    (m*n)
% s ---> storage cost (m*1)
%
% output parameters
% x* ---> optimal solution (m*n)
% c* ---> minimum transportation cost
clc;
clear all;
disp('Enter the input parametrs')
disp('-------------------------')
a = input('Enter the supply, a :')
b = input('Enter the demand, b :')
c = input('Enter the cost values, c:')
s = input('Enter the storage cost, s:')
disp('Input parameters are entered')
disp('----------------------------')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          NORTH WEST CORNER RULE        %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[x,d,c]= NWCR(a,b,c,s);
disp('Initial basic solution ')
disp('x(initial) =')
disp(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        COMPUTATION OF DUAL VARIABLE    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Step-2(a)')
[u,v]=dualvariable(x,d,c);
disp('dual of suplly u =')
disp(u')
disp('dual of demand v =')
disp(v')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   LOCATION OF BASIC ENTERING ELEMENT    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Step-2(b)')
[row,col,Rd]=nonbasic(u,v,c,d);
disp('Reduced cost, Rd =')
disp(Rd)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         UPDATING THE TABLEU        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[x,bout]=cycle(x,row,col,d,c,Rd);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         FINAL OUTPUT       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp(' the optimal solution is')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('x* =')
[m n]=size(x);
for i=1:m
```

```
    for j=1:n
      if(bout(i,j)~=0)
        disp(sprintf('x(%d%d)= %d ',i,j,x(i,j)))
      end
    end
end
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('Minimum transportation cost is')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
sprintf('Z*= %d\n',sum(sum(x.*c)))
```

Flowchart:



Results:

a=[30 80 10 60] b=[10 50 20 80]

c=

```
3  4  6  8
2  2  4  5
2  2  2  3
3  3  2  4
```

S=
```
9
5
2
2
```

Enter the input parameters

Enter the supply, a :[30 80 10 60]

a = 30 80 10 60

Enter the demand, b :[10 50 20 80]

b = 10 50 20 80

Enter the cost values, c:[3 4 6 8;2 2 4 5;2 2 2 3;3 3 2 4]

c =

```
3  4  6  8
2  2  4  5
2  2  2  3
3  3  2  4
```

Enter the storage cost, s:[9 5 2 2]

s = 9 5 2 2

Input parameters are entered

Error: unbalanced transportation problem

Supply is greater than demand, SO add a dummy destination

Dummy destination added b(5)= 20

Balanced network is

180

Initial basic solution

x(initial) =

10	20	0	0	0
0	30	20	30	0
0	0	0	10	0
0	0	0	40	20

Step-2(a)

dual of supply u = 5 3 1 2

dual of demand v = -2 -1 1 2 0

Step-2(b)

Reduced cost, Rd =

0	0	0	1	4
1	0	0	0	2
3	2	0	0	1
3	2	-1	0	0

Element entering the basic

R(4,3)= -1

Step-3

Update Tableau

x(updated) =

10	20	0	0	0
0	30	0	50	0
0	0	0	10	0
0	0	20	20	20

Step-2(a)

dual of suplly u = 5 3 1 2

dual of demand v = -2 -1 0 2 0

Step-2(b)

Reduced cost, Rd =

0	0	1	1	4
1	0	1	0	2
3	2	1	0	1
3	2	0	0	0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

the optimal solution is

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x* =

x(11)= 10

x(12)= 20

x(22)= 30

x(24)= 50

x(34)= 10

x(43)= 20

x(44)= 20

x(45)= 20

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Minimum transportation cost is

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Z*= 610