# EE 309 POWER SYSTEMS

*COURSE PROJECT*

# DG PLACEMENT STUDY - I



| | |
|---|---|
| **ARPIT KUMAR VERMA** | **2021EEB1156** |
| **ARPIT VATS** | **2021EEB1157** |
| **PRIYANSHU NANDA** | **2021EEB1197** |
| **RAHUL** | **2021EEB1199** |
| **AARYAN YASHVARDHAN SHARMA** | **2021EEB1259** |

## Introduction:

This project aims to optimize the placement of distributed generators (DGs) in power systems. The method begins with the application of the forward-backward sweep algorithm on reference data. The output is then fed into the Combined Power Loss Sensitivity function, which selects the top 50% of buses most immune to power loss sensitivity. An objective function is then applied, which comprises three components: F1 for power loss minimization, F2 for total voltage deviation, and F3 for voltage stability. The final step involves the use of an improved Grey Wolf algorithm to determine the most suitable position for DG placement. This comprehensive approach ensures efficient and stable power distribution.

### *1)* **Input Data:** *Standard Data taken from IEEE paper*

```
%bus no, Pload, Qload
BD=[1    0      0      0
    2    100    60     0
    3    90     40     0
    4    120    80     0
    5    60     30     0
    6    60     20     0
    7    200    100    0
    8    200    100    0
    9    60     20     0
    10   60     20     0
    11   45     30     0
    12   60     35     0
    13   60     35     0
    14   120    80     0
    15   60     10     0
    16   60     20     0
    17   60     20     0
    18   90     40     0
    19   90     40     0
    20   90     40     0
    21   90     40     0
    22   90     40     0
    23   90     50     0
    24   420    200    0
    25   420    200    0
    26   60     25     0
    27   60     25     0
    28   60     20     0
    29   120    70     0
    30   200    600    0
    31   150    70     0
    32   210    100    0
    33   60     40     0];
```

Fig – Bus Data

```
%index, starting bus, ending bus, R, X
LD=[1    1    2    0.0922   0.0470
    2    2    3    0.4930   0.2511
    3    3    4    0.3660   0.1864
    4    4    5    0.3811   0.1941
    5    5    6    0.8190   0.7070
    6    6    7    0.1872   0.6188
    7    7    8    0.7114   0.2351
    8    8    9    1.0300   0.7400
    9    9    10   1.0440   0.7400
    10   10   11   0.1966   0.0650
    11   11   12   0.3744   0.1238
    12   12   13   1.4680   1.1550
    13   13   14   0.5416   0.7129
    14   14   15   0.5910   0.5260
    15   15   16   0.7463   0.5450
    16   16   17   1.2890   1.7210
    17   17   18   0.7320   0.5740
    18   2    19   0.1640   0.1565
    19   19   20   1.5042   1.3554
    20   20   21   0.4095   0.4784
    21   21   22   0.7089   0.9373
    22   3    23   0.4512   0.3083
    23   23   24   0.8980   0.7091
    24   24   25   0.8960   0.7011
    25   6    26   0.2030   0.1034
    26   26   27   0.2842   0.1447
    27   27   28   1.0590   0.9337
    28   28   29   0.8042   0.7006
    29   29   30   0.5075   0.2585
    30   30   31   0.9744   0.9630
    31   31   32   0.3105   0.3619
    32   32   33   0.3410   0.5302 ];
```

Fig – Line data

## 2) Combined Power loss Sensitivity:

$$\text{CPLS} = \text{R}\left(\frac{2P_k}{|V_k|^2}\right) + j\text{X}\left(\frac{2Q_k}{|V_k|^2}\right)$$

$V_k$ = voltage of branch k

Pk = real power in branch k

$Q_k$ = reactive power in branch k

R = resistance of branch k

X = reactance of branch k

Performing load flow, we determined CPLS, and calculated potential buses for DGs.
Buses that have high CPLS values are considered candidates for DG installation.
**50% of the buses (16 in this case) were removed from further analysis as their CPLS was low.**

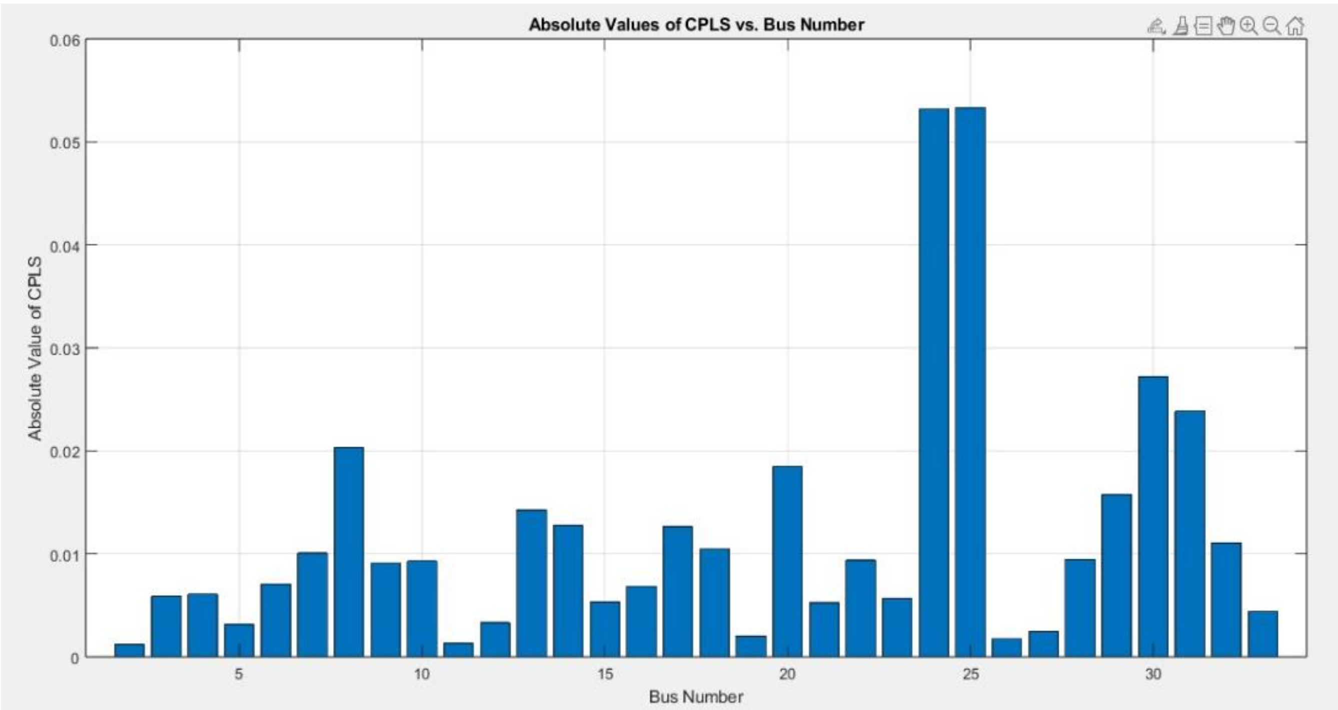



Fig - CPLS values

## 3) Objective function : To ne used in Grey Wolf Optimisation

$$MOF = \mathcal{R}_1 * F_1 + \mathcal{R}_2 * F_2 + \mathcal{R}_3 * F_3$$

Where R1, R2 and R3 are weights, which satisfy:

$$\mathcal{R}_1 + \mathcal{R}_2 + \mathcal{R}_3 = 1$$

Individual Functions:

F1 - The objective function for power loss minimization

$$F_1 = \text{ Min real } \left(\sum_{i=1}^{N} S_{i \text{ Total loss}}\right)$$

F2 - Total of voltage deviation

$$F_2 = TVD = \sum_{n=1}^{NB} (V_n - V_{ref})^2$$

F3 - Objective function for voltage stability

$$F_3 = \frac{1}{\sum_{n=1}^{NB} VSI_{(n)}} \mid$$

Where VSI is calculated by:

$$VSI_{(k)} = |V_m|^4 - 4\left((P_k + P_{k,\text{ Load}})X - (Q_k + Q_{k,\text{ Load}})R\right)^2 - 4$$
$$\left((P_k + P_{k,\text{ Load}})X + (P_k + P_{k,\text{ Load}})R\right)|V_m|^2$$

Vm : represent voltage magnitude in bus m.

PK, Qk : indicate active and reactive powers flowing in bus k, respectively. While Pk, Load , Qk , Load signify real and reactive demands in bus k, respectively.

R.X : resistance and reactance in the branch between buses k and m, respectively.

VSI(k) : indicates to the voltage stability index for at the bus k;

F3 : indicates to the third objective function represented by voltage stability index.

## 4) MATLAB Code:

a) For P type:

```
Sbase=100;                  % BASE MVA VALUE
Vbase=12.66;                   % BASE KV VALUE
Zbase=(Vbase^2)/Sbase;      % BASE IMPEDANCE VALUE
LD(:,4:5)=LD(:,4:5)/Zbase;  % PER UNIT VALUES OF LINE R AND X
BD(:,2:3)=BD(:,2:3)/(1000*Sbase);  % PER UNIT VALUES LOADS P AND Q AT EACH BUS
N=max(max(LD(:,2:3)));

V=ones(size(BD,1),1);                   % INITIAL VALUE OF VOLTAGES
Z=complex(LD(:,4),LD(:,5));     % LINE IMPEDANCE IN PER UNIT VALUES
Iline=zeros(size(LD,1),1);      % LINE CURRENT MATRIX
```

Fig- Selecting the base values for per unit analysis and declaring variables

```matlab
function [Tl, TQl, Voltage, Vangle, Ploss, Qloss] = calculate_power_losses(LD,BD,  Z, V, Sbase, Iter)
    % MAXIMUM NUMBER OF ITERATIONS FOR BACKWARD-FORWARD SWEEP ALGORITHM
    % LD: Load data matrix
    % Z: Line impedance vector
    % V: Initial voltage vector
    % Sbase: System base power
    % Iter: Maximum number of iterations

    Sload=complex(BD(:,2),BD(:,3)); % COMPLEX LOAD IN PER UNIT VALUES
    % Initialize variables
    Iload = zeros(size(V)); % Assuming Iload is initialized to zero
    Iline = zeros(size(LD, 1), 1); % Initialize line current vector

    for i = 1:Iter
        % STARTING WITH BACKWARD SWEEP FOR LINE CURRENT CALCULATIONS
        Iload = conj(Sload./V);
        for j = size(LD,1):-1:1   % STARTING FROM THE END OF THE FEEDER
            c = [];
            e = [];
            [c e] = find(LD(:,2:3)==LD(j,3));
            if size(c,1)==1     % IF SIZE(C,1) IS ONE THAN BUS "J" IS STARTING OR ENDING BUS
                Iline(LD(j,1)) = Iload(LD(j,3));
            else
                Iline(LD(j,1)) = Iload(LD(j,3)) + sum(Iline(LD(c,1))) - Iline(LD(j,1));
            end
        end
        % STARTING THE FORWARD SWEEP FOR BUS-VOLTAGE CALCULATION
        for j = 1:size(LD,1)
            V(LD(j,3)) = V(LD(j,2)) - Iline(LD(j,1))*Z(j);
        end
    end

    % Calculate power losses and voltages
    Voltage = abs(V);
    Vangle = angle(V);
    Ploss = real(Z.*(abs(Iline.^2)));
    Qloss = imag(Z.*(abs(Iline.^2)));
    Tl = sum(Ploss)*Sbase*1000;
    TQl = sum(Qloss)*Sbase*1000;
end
```

Fig – Power loss calculation function

```matlab
absCPLS_values = zeros(32, 1); % Assuming 33 buses, but we start from 2, so 32 values

% Initialize a vector to store the bus numbers
bus_numbers = 2:33; % Bus numbers from 2 to 33

for x = 2:33
    curvol = Voltage(x,1);
    curpowerreal = BD(x,2);
    curpowerim = BD(x,3);
    curR = LD(x-1,4);
    curX = LD(x-1,5);

    curCPLS = (2*curpowerreal*curR)/(curvol^2) + 1i*(2*curpowerim*curX)/(curvol^2);
    absCPLS = abs(curCPLS); % Calculate the absolute value

    % Store the absolute value in the vector
    absCPLS_values(x-1) = absCPLS;
end
```

Fig- Calculate the CPLS values for buses

```
function OBJ = calculateObjectiveFunction(Voltage, BD, LD, Ploss,Qloss)
    TPL = 0;
    TVD = 0;
    VSI = 0;
    for x = 2:33
        curvol = Voltage(x,1);
        curpowerreal = BD(x,2);
        curpowerim = BD(x,3);
        curR = LD(x-1,4);
        curX = LD(x-1,5);

        TPL = TPL + Ploss(x-1,1);

        TVD = TVD + ((curvol - Voltage(1,1))^2);

        VSI = VSI + curvol^4 - (4*(curpowerreal*curR + curpowerim*curX)*curvol^2) - (4*(curpowerreal*curX + curpowerim*curR));
    end
    complexVector = sum(Ploss) + 1i*sum(Qloss); % Combine real and imaginary parts
    s_loss = norm(complexVector);
    F1 = s_loss;
    F2 = TVD;
    F3 = 1/VSI;
    R1 = 0.8;
    R2 = 0.1;
    R3 = 0.1;
    OBJ = F1*R1 + F2*R2 + F3*R3;
end



OBJ = calculateObjectiveFunction(Voltage, BD, LD, Ploss,Qloss);
disp(OBJ);
```

Fig – Calculate the objective function for buses

```
% Define the range of sizes
no_of_dg_avail = 128;
sizes_range = linspace(1500/(1000*Sbase), 3000/(1000*Sbase), no_of_dg_avail);

% Initialize a cell array to store the tuples
bus_sizes_matrix = cell(length(selected_buses), length(sizes_range));

% Fill the cell array with tuples of bus numbers and sizes
for i = 1:length(selected_buses)
    for j = 1:length(sizes_range)
        % Create a tuple (bus number, size) and store it in the cell array
        bus_sizes_matrix{i, j} = [selected_buses(i), sizes_range(j)];
    end
end

% Display the cell array
disp(bus_sizes_matrix)
```

Fig – Initialise the bus size matrix for all possible combinations of bus number and DG size

```
num_iter = 50;
num_agents = 64;


% Initialize an empty cell array to store the tuples
location_agent = cell(1, num_agents);

% Generate 50 tuples (i, j) with i and j being random integers between 1 and 16
for k = 1:num_agents
    i = randi([1, 16]); % Random integer between 1 and 16
    j = randi([1, 16]); % Random integer between 1 and 16
    location_agent{k} = [i, j]; % Store the tuple in the cell array
end

% Display the cell array of tuples
disp(location_agent);
```

Fig – Assigning random tuples to the agents.

```
%1-location, size,obj,agentnumber
alpha = zeros(4,1);
beta = zeros(4,1);
delta = zeros(4,1);

alpha(3)=intmax('int32');
beta(3)=intmax('int32');
delta(3)=intmax('int32');
function [Z_t_1_alpha, Z_t_1_alpha_indices] = search(bus_sizes_matrix, Z_following_alpha)
    % Initialize variables to store the closest match and its index
    closestMatch = [];
    closestIndex = [];
    minDifference = inf; % Initialize with infinity to ensure any first comparison will be smaller

    % Iterate through each cell in bus_sizes_matrix
    for i = 1:size(bus_sizes_matrix, 1)
        for j = 1:size(bus_sizes_matrix, 2)
            % Extract the numeric array from the cell
            currentElement = bus_sizes_matrix{i, j};

            % Calculate the difference between the current element and Z_following_alpha
            difference = norm(currentElement - Z_following_alpha);

            % Check if the current difference is smaller than the previous minimum difference
            if difference < minDifference
                % Update the closest match and its index
                closestMatch = currentElement;
                closestIndex = [i, j];
                minDifference = difference;
            end
        end
    end
```

Fig – Discretising the values generated(to avoid skipping cases)

```
for cur_agent = 1:num_agents
        % Assuming location_agent is a cell array where each cell contains a 2-element vector
        % representing the row and column indices in BD to update
        % location_agent{cur_agent}
        i = location_agent{cur_agent}(1);
        j = location_agent{cur_agent}(2);
        agent_loc_size = bus_sizes_matrix(i,j);
        bus_for_DG = agent_loc_size{1}(1);
        dg_size = agent_loc_size{1}(2);
        % dg_size = location_agent{cur_agent}(2);
        BD_temp = BD;
       % Store the old load power
        old_Pload = BD(bus_for_DG, 2);

        % Update the load power based on the new value
        % Assuming the new value is stored in the second column of location_agent
       BD_temp(bus_for_DG, 2) = old_Pload - dg_size;
       [Tl, TQl, Voltage, Vangle, Ploss, Qloss] = calculate_power_losses(LD,BD_temp,  Z, V, Sbase, num_iter/10);
       agent_obj =calculateObjectiveFunction(Voltage, BD_temp, LD, Ploss,Qloss);


        if agent_obj<alpha(3)
            alpha(1)=bus_for_DG;
            alpha(2)=dg_size;
            alpha(3)=agent_obj;
            alpha(4)=cur_agent;
        elseif agent_obj<beta(3)
            beta(1)=bus_for_DG;
            beta(2)=dg_size;
            beta(3)=agent_obj;
            beta(4)=cur_agent;
        elseif agent_obj<delta(3)
            delta(1)=bus_for_DG;
            delta(2)=dg_size;
            delta(3)=agent_obj;
            delta(4)=cur_agent;
        end
```

Fig- Performing load flow for each agent and assigning the alpha, beta and delta wolf

```
r1 = rand;
r2 = rand;
C=2*r2;
A=2*a*r1 - a;

i_t = location_agent{cur_agent}(1);
j_t = location_agent{cur_agent}(2);
agent_loc_size = bus_sizes_matrix(i_t,j_t);
bus_for_DG = agent_loc_size{1}(1);
dg_size = agent_loc_size{1}(2);

Z_t = [bus_for_DG dg_size];
D_alpha=abs(C*Z_alpha - Z_t);
D_beta = abs(C*Z_beta - Z_t);
D_delta = abs(C*Z_delta - Z_t);

Z_following_alpha= Z_alpha- A*D_alpha;
Z_following_beta= Z_beta- A*D_beta;
Z_following_delta= Z_delta- A*D_delta;

[Z_t_1_alpha,  Z_t_1_alpha_indices]= search(bus_sizes_matrix,Z_following_alpha);
[Z_t_1_beta, Z_t_1_beta_indices] = search(bus_sizes_matrix,Z_following_beta);
[Z_t_1_delta, Z_t_1_delta_indices] = search(bus_sizes_matrix,Z_following_delta);

Z_final_temp = (Z_t_1_alpha + Z_t_1_beta + Z_t_1_delta)/3;
[Z_final, Z_final_indices] = search(bus_sizes_matrix,Z_final_temp);
```

Fig – Calculating the next position for the wolves using GWO

```
%DLH

del_i=abs(i_t - i_t_1);
del_j=abs(j_t - j_t_1);

i_neigh_gwo = randi([abs(i_t-del_i), abs(i_t+del_i)]);
j_neigh_gwo = randi([abs(j_t-del_j), abs(j_t+del_j)]);

i_neigh_pop = randi([1, length(selected_buses)]);
j_neigh_pop = randi([1, no_of_dg_avail]);

Z_DLH_indices = [0 0];


Z_DLH_indices(1) = randi([ abs(i_t-abs(i_neigh_pop- i_neigh_gwo)), abs(i_t+abs(i_neigh_pop- i_neigh_gwo))]);
Z_DLH_indices(2) = randi([ abs(j_t-abs(j_neigh_pop- j_neigh_gwo)), abs(j_t+abs(j_neigh_pop- j_neigh_gwo))]);
if Z_DLH_indices(1) > length(selected_buses)
    Z_DLH_indices(1) = length(selected_buses);
end
if Z_DLH_indices(2) > no_of_dg_avail
    Z_DLH_indices(2) = no_of_dg_avail;
end
if Z_DLH_indices(1) < 1
    Z_DLH_indices(1) = 1;
end
if Z_DLH_indices(2) < 1
    Z_DLH_indices(2) = 1;
end

%agent_loc_size = bus_sizes_matrix(i_t,j_t);
Z_DLH = bus_sizes_matrix(Z_DLH_indices(1),Z_DLH_indices(2));
bus_for_DG_DLH = Z_DLH{1}(1);
dg_size_DLH = Z_DLH{1}(2);

BD_temp = BD;
```

Fig- Performing the DLH approach

```
 % Update the load power based on the new value
 % Assuming the new value is stored in the second column of location_agent
BD_temp(bus_for_DG_DLH, 2) = old_Pload - dg_size_DLH;
[Tl, TQl, Voltage_DLH, Vangle, Ploss_DLH, Qloss_DLH] = calculate_power_losses(LD,BD_temp,  Z, V, Sbase, num_iter/10);
agent_obj_DLH =calculateObjectiveFunction(Voltage_DLH, BD_temp, LD, Ploss_DLH, Qloss_DLH);

agent_obj=agent_obj_GWO;
 bus_for_DG= bus_for_DG_GWO;
 dg_size= dg_size_GWO;
 location_agent{cur_agent}=Z_final_indices;
 Ploss=Ploss_GWO;
 Qloss=Qloss_GWO;
 Voltage=Voltage_GWO;

if agent_obj_GWO>agent_obj_DLH
      agent_obj=agent_obj_DLH;
      bus_for_DG= bus_for_DG_DLH;
      dg_size= dg_size_DLH;
      location_agent{cur_agent}=Z_DLH_indices;
      Ploss=Ploss_DLH;
      Qloss=Qloss_DLH;
      Voltage=Voltage_DLH;
end
```

Fig – Updating the load power with optimised value

```
for i=1:33
     if Voltage(i)>1.1 || Voltage(i)<0.9
         constraint_broken=1;
     end
 end

  if constraint_broken==1
      continue;
  end

 location_agent{cur_agent}=Z_final_indices;


 if agent_obj<alpha(3)
     alpha(1)=bus_for_DG;
     alpha(2)=dg_size;
     alpha(3)=agent_obj;
     alpha(4)=cur_agent;
     complexVector = sum(Ploss) + 1i*sum(Qloss); % Combine real and imaginary parts
     final_loss= norm(complexVector); % Calculate the magnitude
     final_Voltage=Voltage;

 elseif agent_obj<beta(3)
     beta(1)=bus_for_DG;
     beta(2)=dg_size;
     beta(3)=agent_obj;
     beta(4)=cur_agent;
 elseif agent_obj<delta(3)
     delta(1)=bus_for_DG;
     delta(2)=dg_size;
     delta(3)=agent_obj;
     delta(4)=cur_agent;
 end
```

Fig – Check constraints and update the results

b) For PQ type:

```
Sbase=100;                  % BASE MVA VALUE
Vbase=12.66;                   % BASE KV VALUE
Zbase=(Vbase^2)/Sbase;      % BASE IMPEDANCE VALUE
LD(:,4:5)=LD(:,4:5)/Zbase;  % PER UNIT VALUES OF LINE R AND X
BD(:,2:3)=BD(:,2:3)/(1000*Sbase);  % PER UNIT VALUES LOADS P AND Q AT EACH BUS
N=max(max(LD(:,2:3)));

V=ones(size(BD,1),1);                    % INITIAL VALUE OF VOLTAGES
Z=complex(LD(:,4),LD(:,5));      % LINE IMPEDANCE IN PER UNIT VALUES
Iline=zeros(size(LD,1),1);       % LINE CURRENT MATRIX
```

Fig- Selecting the base values for per unit analysis and declaring variables

```
function [Tl, TQl, Voltage, Vangle, Ploss, Qloss] = calculate_power_losses(LD,BD,  Z, V, Sbase, Iter)
    % MAXIMUM NUMBER OF ITERATIONS FOR BACKWARD-FORWARD SWEEP ALGORITHM
    % LD: Load data matrix
    % Z: Line impedance vector
    % V: Initial voltage vector
    % Sbase: System base power
    % Iter: Maximum number of iterations

    Sload=complex(BD(:,2),BD(:,3)); % COMPLEX LOAD IN PER UNIT VALUES
    % Initialize variables
    Iload = zeros(size(V)); % Assuming Iload is initialized to zero
    Iline = zeros(size(LD, 1), 1); % Initialize line current vector

    for i = 1:Iter
        % STARTING WITH BACKWARD SWEEP FOR LINE CURRENT CALCULATIONS
        Iload = conj(Sload./V);
        for j = size(LD,1):-1:1   % STARTING FROM THE END OF THE FEEDER
            c = [];
            e = [];
            [c e] = find(LD(:,2:3)==LD(j,3));
            if size(c,1)==1     % IF SIZE(C,1) IS ONE THAN BUS "J" IS STARTING OR ENDING BUS
                Iline(LD(j,1)) = Iload(LD(j,3));
            else
                Iline(LD(j,1)) = Iload(LD(j,3)) + sum(Iline(LD(c,1))) - Iline(LD(j,1));
            end
        end
        % STARTING THE FORWARD SWEEP FOR BUS-VOLTAGE CALCULATION
        for j = 1:size(LD,1)
            V(LD(j,3)) = V(LD(j,2)) - Iline(LD(j,1))*Z(j);
        end
    end

    % Calculate power losses and voltages
    Voltage = abs(V);
    Vangle = angle(V);
    Ploss = real(Z.*(abs(Iline.^2)));
    Qloss = imag(Z.*(abs(Iline.^2)));
    Tl = sum(Ploss)*Sbase*1000;
    TQl = sum(Qloss)*Sbase*1000;
end
```

Fig – Power loss calculation function

```
absCPLS_values = zeros(32, 1); % Assuming 33 buses, but we start from 2, so 32 values

% Initialize a vector to store the bus numbers
bus_numbers = 2:33; % Bus numbers from 2 to 33

for x = 2:33
    curvol = Voltage(x,1);
    curpowerreal = BD(x,2);
    curpowerim = BD(x,3);
    curR = LD(x-1,4);
    curX = LD(x-1,5);

    curCPLS = (2*curpowerreal*curR)/(curvol^2) + 1i*(2*curpowerim*curX)/(curvol^2);
    absCPLS = abs(curCPLS); % Calculate the absolute value

    % Store the absolute value in the vector
    absCPLS_values(x-1) = absCPLS;
end
```

Fig- Calculate the CPLS values for buses

```
function OBJ = calculateObjectiveFunction(Voltage, BD, LD, Ploss,Qloss)
    TPL = 0;
    TVD = 0;
    VSI = 0;
    for x = 2:33
        curvol = Voltage(x,1);
        curpowerreal = BD(x,2);
        curpowerim = BD(x,3);
        curR = LD(x-1,4);
        curX = LD(x-1,5);

        TPL = TPL + Ploss(x-1,1);

        TVD = TVD + ((curvol - Voltage(1,1))^2);

        VSI = VSI + curvol^4 - (4*(curpowerreal*curR + curpowerim*curX)*curvol^2) - (4*(curpowerreal*curX + curpowerim*curR));
    end
    complexVector = sum(Ploss) + 1i*sum(Qloss); % Combine real and imaginary parts
    s_loss = norm(complexVector);
    F1 = s_loss;
    F2 = TVD;
    F3 = 1/VSI;
    R1 = 0.8;
    R2 = 0.1;
    R3 = 0.1;
    OBJ = F1*R1 + F2*R2 + F3*R3;
end



OBJ = calculateObjectiveFunction(Voltage, BD, LD, Ploss,Qloss);
disp(OBJ);
```

Fig – Calculate the objective function for buses

```
% Define the range of sizes
no_of_dg_avail = 16;
sizes_range_P = linspace(1500/(1000*Sbase), 3000/(1000*Sbase), no_of_dg_avail);
sizes_range_Q = linspace(1500/(1000*Sbase), 3000/(1000*Sbase), no_of_dg_avail);

% Initialize a cell array to store the tuples
bus_sizes_matrix = cell(length(selected_buses), length(sizes_range_P), length(sizes_range_Q));

% Fill the cell array with tuples of bus numbers and sizes
for i = 1:length(selected_buses)
    for j = 1:length(sizes_range_P)
        for k = 1:length(sizes_range_Q)
        % Create a tuple (bus number, size) and store it in the cell array
            bus_sizes_matrix{i, j, k} = [selected_buses(i), sizes_range_P(j), sizes_range_Q(k)];
        end
    end
end

% Display the cell array
disp(bus_sizes_matrix);
```

Fig – Initialise the bus size matrix for all possible combinations of bus number and DG size

```
num_iter = 5;
num_agents = 64;


% Initialize an empty cell array to store the tuples
location_agent = cell(1, num_agents);

% Generate 50 tuples (i, j) with i and j being random integers between 1 and 16
for cur = 1:num_agents
    i = randi([1, length(selected_buses)]); % Random integer between 1 and 16
    j = randi([1, no_of_dg_avail]); % Random integer between 1 and 16
    k = randi([1, no_of_dg_avail]); % Random integer between 1 and 16
    location_agent{cur} = [i, j, k]; % Store the tuple in the cell array
end

% Display the cell array of tuples
disp(location_agent);
```

Fig – Assigning random tuples to the agents.

```
%1-location, p_size, q_size,obj,agentnumber
alpha = zeros(5,1);
beta = zeros(5,1);
delta = zeros(5,1);

alpha(4)=intmax('int32');
beta(4)=intmax('int32');
delta(4)=intmax('int32');


function [Z_t_1_alpha, Z_t_1_alpha_indices] = search(bus_sizes_matrix, Z_following_alpha)
    % Initialize variables to store the closest match and its index
    closestMatch = [];
    closestIndex = [];
    minDifference = inf; % Initialize with infinity to ensure any first comparison will be smaller

    % Iterate through each cell in bus_sizes_matrix
    for i = 1:size(bus_sizes_matrix, 1)
        for j = 1:size(bus_sizes_matrix, 2)
            for k = 1:size(bus_sizes_matrix, 3)
                % Extract the numeric array from the cell
                currentElement = bus_sizes_matrix{i, j, k};

                % Calculate the difference between the current element and Z_following_alpha
                difference = norm(currentElement - Z_following_alpha);

                % Check if the current difference is smaller than the previous minimum difference
                if difference < minDifference
                    % Update the closest match and its index
                    closestMatch = currentElement;
                    closestIndex = [i, j, k];
                    minDifference = difference;
                end
            end
        end
    end
```

Fig – Discretising the values generated(to avoid skipping cases)

```
for cur_agent = 1:num_agents
        % Assuming location_agent is a cell array where each cell contains a 2-element vector
        % representing the row and column indices in BD to update
        % location_agent{cur_agent}
        i = location_agent{cur_agent}(1);
        j = location_agent{cur_agent}(2);
        k = location_agent{cur_agent}(3);
        agent_loc_size = bus_sizes_matrix(i,j,k);
        bus_for_DG = agent_loc_size{1}(1);
        dg_size_p = agent_loc_size{1}(2);
        dg_size_q = agent_loc_size{1}(3);
        % dg_size = location_agent{cur_agent}(2);
        BD_temp = BD;
       % Store the old load power
        old_Pload = BD(bus_for_DG, 2);
        old_Qload = BD(bus_for_DG, 3);
        % Update the load power based on the new value
        % Assuming the new value is stored in the second column of location_agent
       BD_temp(bus_for_DG, 2) = old_Pload - dg_size_p;
       BD_temp(bus_for_DG, 3) = old_Qload - dg_size_q;
       [Tl, TQl, Voltage, Vangle, Ploss, Qloss] = calculate_power_losses(LD,BD_temp,  Z, V, Sbase, num_iter);
        Ploss;
        Qloss;
       agent_obj =calculateObjectiveFunction(Voltage, BD_temp, LD, Ploss,Qloss);

        if agent_obj<alpha(4)
            alpha(1)=bus_for_DG;
            alpha(2)=dg_size_p;
            alpha(3)=dg_size_q;
            alpha(4)=agent_obj;
            alpha(5)=cur_agent;
        elseif agent_obj<beta(4)
            beta(1)=bus_for_DG;
            beta(2)=dg_size_p;
            beta(3)=dg_size_q;
            beta(4)=agent_obj;
            beta(5)=cur_agent;
        elseif agent_obj<delta(4)
            delta(1)=bus_for_DG;
            delta(2)=dg_size_p;
            delta(3)=dg_size_q;
            delta(4)=agent_obj;
            delta(5)=cur_agent;
        end
```

Fig- Performing load flow for each agent and assigning the alpha, beta and delta wolf

```
r1 = rand;
r2 = rand;
C=2*r2;
A=2*a*r1 - a;

i_t = location_agent{cur_agent}(1);
j_t = location_agent{cur_agent}(2);
k_t = location_agent{cur_agent}(3);
agent_loc_size = bus_sizes_matrix(i_t,j_t,k_t);
bus_for_DG = agent_loc_size{1}(1);
dg_size_p = agent_loc_size{1}(2);
dg_size_q = agent_loc_size{1}(3);

Z_t = [bus_for_DG dg_size_p dg_size_q];
D_alpha=abs(C*Z_alpha - Z_t);
D_beta = abs(C*Z_beta - Z_t);
D_delta = abs(C*Z_delta - Z_t);

Z_following_alpha= abs(Z_alpha- A*D_alpha);
Z_following_beta= abs(Z_beta- A*D_beta);
Z_following_delta= abs(Z_delta- A*D_delta);

[Z_t_1_alpha,  Z_t_1_alpha_indices]= search(bus_sizes_matrix,Z_following_alpha);
[Z_t_1_beta, Z_t_1_beta_indices] = search(bus_sizes_matrix,Z_following_beta);
[Z_t_1_delta, Z_t_1_delta_indices] = search(bus_sizes_matrix,Z_following_delta);

Z_final_temp = (Z_t_1_alpha + Z_t_1_beta + Z_t_1_delta)/3;
[Z_final, Z_final_indices] = search(bus_sizes_matrix,Z_final_temp);
```

Fig – Calculating the next position for the wolves using GWO

```
%DLH

del_i=abs(i_t - i_t_1);
del_j=abs(j_t - j_t_1);
del_k=abs(k_t - k_t_1);
i_neigh_gwo = randi([abs(i_t-del_i), abs(i_t+del_i)]);
j_neigh_gwo = randi([abs(j_t-del_j), abs(j_t+del_j)]);
k_neigh_gwo = randi([abs(k_t-del_k), abs(k_t+del_k)]);

i_neigh_pop = randi([1, length(selected_buses)]);
j_neigh_pop = randi([1, no_of_dg_avail]);
k_neigh_pop = randi([1, no_of_dg_avail]);

Z_DLH_indices = [0 0 0];

Z_DLH_indices(1) = randi([ abs(i_t-abs(i_neigh_pop- i_neigh_gwo)), abs(i_t+abs(i_neigh_pop- i_neigh_gwo))]);
Z_DLH_indices(2) = randi([ abs(j_t-abs(j_neigh_pop- j_neigh_gwo)), abs(j_t+abs(j_neigh_pop- j_neigh_gwo))]);
Z_DLH_indices(3) = randi([ abs(k_t-abs(k_neigh_pop- k_neigh_gwo)), abs(k_t+abs(k_neigh_pop- k_neigh_gwo))]);
if Z_DLH_indices(1) > length(selected_buses)
    Z_DLH_indices(1) = length(selected_buses);
end
if Z_DLH_indices(2) > no_of_dg_avail
    Z_DLH_indices(2) = no_of_dg_avail;
end
if Z_DLH_indices(1) < 1
    Z_DLH_indices(1) = 1;
end
if Z_DLH_indices(2) < 1
    Z_DLH_indices(2) = 1;
end
if Z_DLH_indices(3) < 1
    Z_DLH_indices(3) = 1;
end
if Z_DLH_indices(3) > no_of_dg_avail
    Z_DLH_indices(3) = no_of_dg_avail;
end
%agent_loc_size = bus_sizes_matrix(i_t,j_t);
Z_DLH = bus_sizes_matrix(Z_DLH_indices(1),Z_DLH_indices(2), Z_DLH_indices(3));
bus_for_DG_DLH = Z_DLH{1}(1);
dg_size_DLH_p = Z_DLH{1}(2);
dg_size_DLH_q = Z_DLH{1}(3);

BD_temp = BD;
```

Fig- Performing the DLH approach

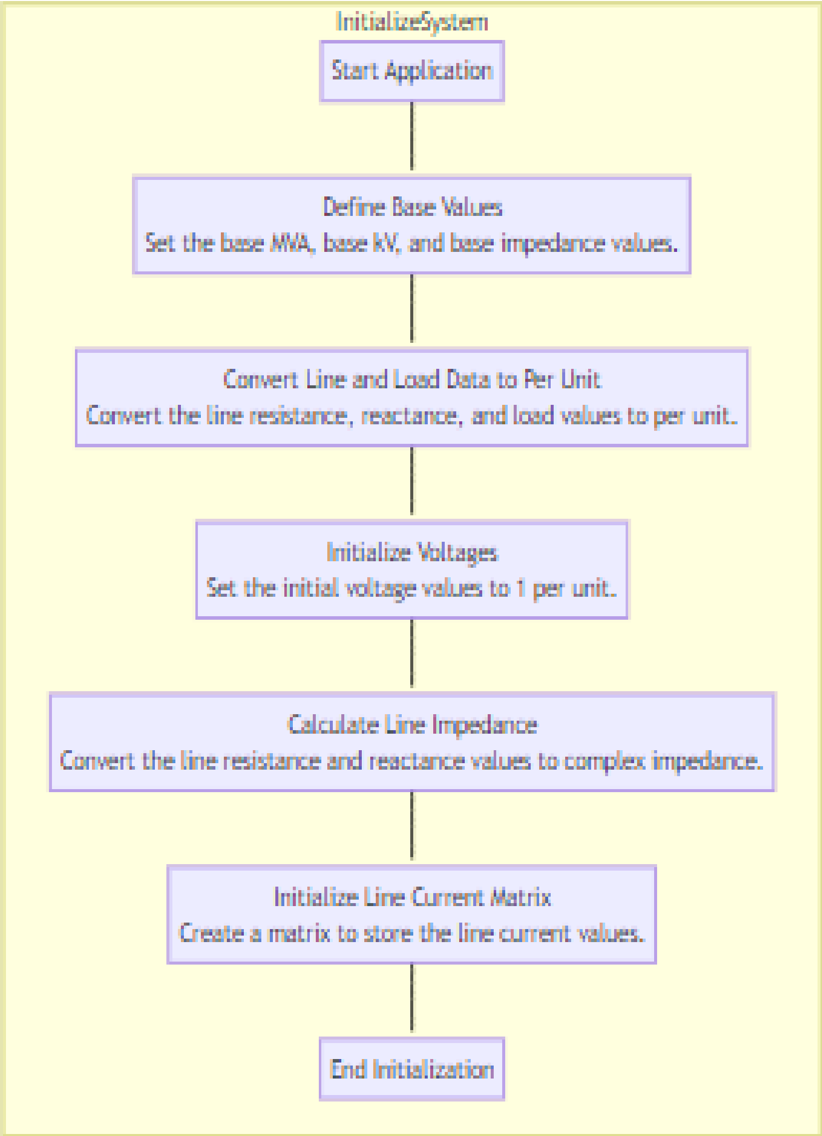
```
% Update the load power based on the new value
% Assuming the new value is stored in the second column of location_agent
BD_temp(bus_for_DG_DLH, 2) = old_Pload - dg_size_DLH_p;
BD_temp(bus_for_DG_DLH, 3) = old_Qload - dg_size_DLH_q;
[T1, TQ1, Voltage_DLH, Vangle, Ploss_DLH, Qloss_DLH] = calculate_power_losses(LD,BD_temp,  Z, V, Sbase, num_iter);
agent_obj_DLH =calculateObjectiveFunction(Voltage_DLH, BD_temp, LD, Ploss_DLH, Qloss_DLH);

agent_obj=agent_obj_GWO;
 bus_for_DG= bus_for_DG_GWO;
 dg_size_p= dg_size_GWO_p;
 dg_size_q= dg_size_GWO_q;
 location_agent{cur_agent}=Z_final_indices;
 Ploss=Ploss_GWO;
 Qloss=Qloss_GWO;
 Voltage=Voltage_GWO;

if agent_obj_GWO>agent_obj_DLH
     agent_obj=agent_obj_DLH;
     bus_for_DG= bus_for_DG_DLH;
     dg_size_p= dg_size_DLH_p;
     dg_size_q= dg_size_DLH_q;
     location_agent{cur_agent}=Z_DLH_indices;
     Ploss=Ploss_DLH;
     Qloss=Qloss_DLH;
     Voltage=Voltage_DLH;
end
```

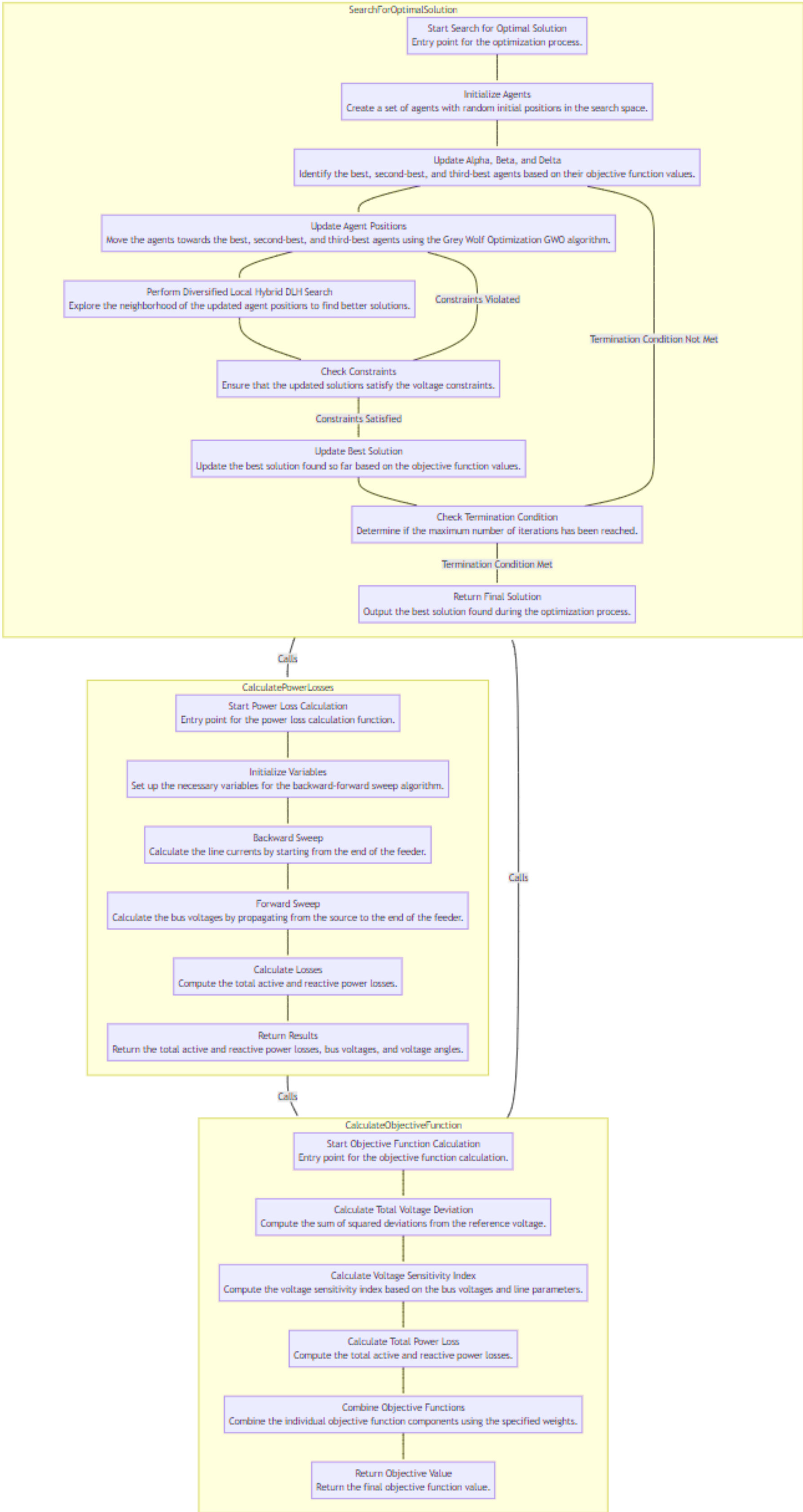
Fig – Updating the load power with optimised value

```
%constraint
constraint_broken=0;
for i=1:33
     if Voltage(i)>1.1 || Voltage(i)<0.9
         constraint_broken=1;
     end
 end

  if constraint_broken==1
      continue;
  end

 location_agent{cur_agent}=Z_final_indices;

 if agent_obj<alpha(4)
     alpha(1)=bus_for_DG;
     alpha(2)=dg_size_p;
     alpha(3)=dg_size_q;
     alpha(4)=agent_obj;
     alpha(5)=cur_agent;
     complexVector = sum(Ploss) + 1i*sum(Qloss); % Combine real and imaginary parts
     final_loss= norm(complexVector); % Calculate the magnitude
     final_Voltage=Voltage;

 elseif agent_obj<beta(4)
     beta(1)=bus_for_DG;
     beta(2)=dg_size_p;
     beta(3)=dg_size_q;
     beta(4)=agent_obj;
     beta(5)=cur_agent;
 elseif agent_obj<delta(4)
     delta(1)=bus_for_DG;
     delta(2)=dg_size_p;
     delta(3)=dg_size_q;
     delta(4)=agent_obj;
     delta(5)=cur_agent;
 end
```

Fig – Check constraints and update the results

# Flow chart of MATLAB code :




SearchForOptimalSolution
Start Search for Optimal Solution
Entry point for the optimization process.
Initialize Agents
Create a set of agents with random initial positions in the search space.
Update Alpha, Beta, and Delta
Identify the best, second-best, and third-best agents based on their objective function values.
Update Agent Positions
Move the agents towards the best, second-best, and third-best agents using the Grey Wolf Optimization GWO algorithm.
Perform Diversified Local Hybrid DLH Search
Explore the neighborhood of the updated agent positions to find better solutions.
Constraints Violated
Termination Condition Not Met
Check Constraints
Ensure that the updated solutions satisfy the voltage constraints.
Constraints Satisfied
Update Best Solution
Update the best solution found so far based on the objective function values.
Check Termination Condition
Determine if the maximum number of iterations has been reached.
Termination Condition Met
Return Final Solution
Output the best solution found during the optimization process.
Calls
CalculatePowerLosses
Start Power Loss Calculation
Entry point for the power loss calculation function.
Initialize Variables
Set up the necessary variables for the backward-forward sweep algorithm.
Backward Sweep
Calculate the line currents by starting from the end of the feeder.
Forward Sweep
Calculate the bus voltages by propagating from the source to the end of the feeder.
Calculate Losses
Compute the total active and reactive power losses.
Return Results
Return the total active and reactive power losses, bus voltages, and voltage angles.
Calls
Calls
CalculateObjectiveFunction
Start Objective Function Calculation
Entry point for the objective function calculation.
Calculate Total Voltage Deviation
Compute the sum of squared deviations from the reference voltage.
Calculate Voltage Sensitivity Index
Compute the voltage sensitivity index based on the bus voltages and line parameters.
Calculate Total Power Loss
Compute the total active and reactive power losses.
Combine Objective Functions
Combine the individual objective function components using the specified weights.
Return Objective Value
Return the final objective function value.

## ***Code explanation:***

MATLAB code aims to optimize Distributed Generation (DG) placement in a power distribution network using an improved version of the Grey Wolf Optimizer (GWO). The GWO is a metaheuristic optimization algorithm inspired by the leadership hierarchy and hunting mechanism of grey wolves in nature. This code enhances the GWO by incorporating a dynamic leader selection mechanism and a diversity preservation strategy to improve the convergence speed and solution quality.

## Initialization

- clc; clear all; close all**;**: Clears the command window, workspace, and closes all figures to start fresh.
- nbus=33: Defines the number of buses in the power system.
- LD and BD: These matrices contain information about the lines and buses, including their connections, resistances (R), reactances (X), loads (Pload, Qload), and other relevant parameters.
- Sbase, Vbase, Zbase: Base values for power, voltage, and impedance, used for per-unit calculations.

## Objective Function Calculation

- Calculate Objective Function: This function calculates the objective function value, which is a combination of power loss, voltage deviation, and voltage stability index (VSI). It aims to minimize these factors to optimize DG placement.

## Load Flow

- Calculate power losses: Calculates the power losses (active and reactive) in the power system due to line impedances and load currents. It uses a backward-forward sweep method for line current calculation and forward sweep for bus voltage calculation.

## CPLS Calculation

- Calculate CPLS (Complex Power Loss Sensitivity): For each bus, it calculates the CPLS, which is a measure of how changes in load power affect the power loss. This helps in identifying critical buses for DG placement.

## Objective Function Optimization

- Calculate Objective Function: This function calculates the objective function value, which is a combination of power loss, voltage deviation, and voltage stability index (VSI). It aims to minimize these factors to optimize DG placement.

## DG Placement Optimization

- Grey Wolf Optimizer (GWO): The main part of the code implements the GWO algorithm to optimize DG placement. It involves:
- Initialization: Setting up the initial population of agents (potential solutions) and defining the objective function.
- Search for the Best Solution: Each agent searches for the best solution in its vicinity, updating its position based on the best, α; second best, β; and third best, δ solutions found so far.
- Dynamic Leader Selection: The leaders (α, β, δ) are dynamically updated based on the fitness of the agents, ensuring that the search process is guided towards better solutions.
- Diversity Preservation: To maintain diversity in the population and prevent premature convergence, a diversity preservation strategy is incorporated.
- Constraint Handling: Checks if the voltage constraints are satisfied after each iteration. If not, the agent continues without updating its position.

## Key Enhancements(IGWO)

- Dynamic Leader Selection: The I-GWO algorithm dynamically updates the positions of the leaders based on the best, second-best, and third-best solutions found so far. This involves modifying the loop where the positions of the agents (wolves) are updated.

- Dimension Learning-Based Hunting (DLBH) Strategy: This strategy involves constructing neighborhoods for each agent and sharing neighboring information. It's crucial for implementing the DLBH strategy in the section where the positions of the agents are updated.

- Diversity Preservation: To prevent premature convergence and maintain diversity in the population, the I-GWO algorithm includes checks to ensure that the solutions found do not cluster too closely together. This can be integrated into the loop where the positions of the agents are updated.

- Constraint Handling: The I-GWO algorithm should also include a mechanism to handle constraints effectively, ensuring that the solutions found comply with the problem's constraints. This involves checking the voltage levels at all buses after each iteration and modifying the solutions if necessary.

- Iterative Updates: The I-GWO algorithm iteratively updates the positions of the agents based on the dynamic leader selection, DLBH strategy, and diversity preservation mechanisms. This iterative process continues until a stopping criterion is met, such as reaching a maximum number of iterations or achieving a satisfactory level of fitness.

# Outputs:

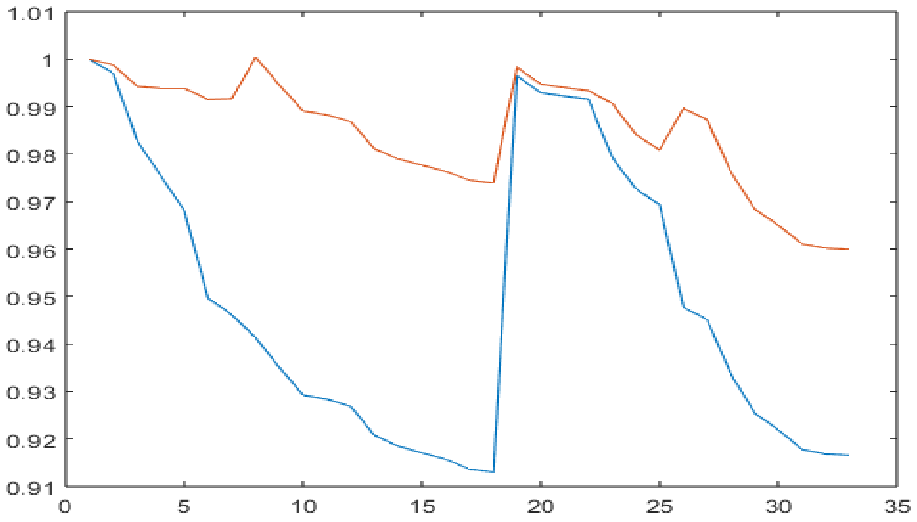
## a) P-type DG:

Voltage Profile :

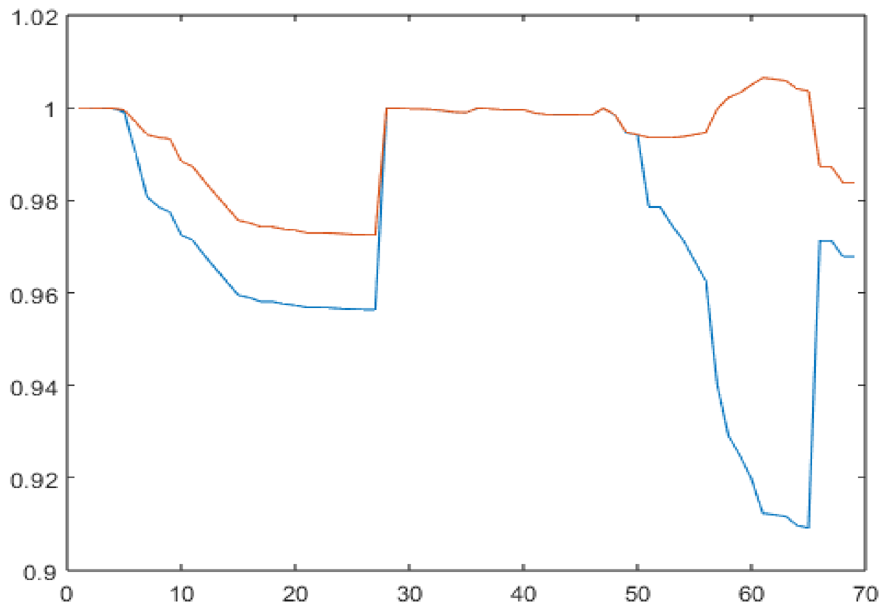
Blue : Without DG

Red : Improved

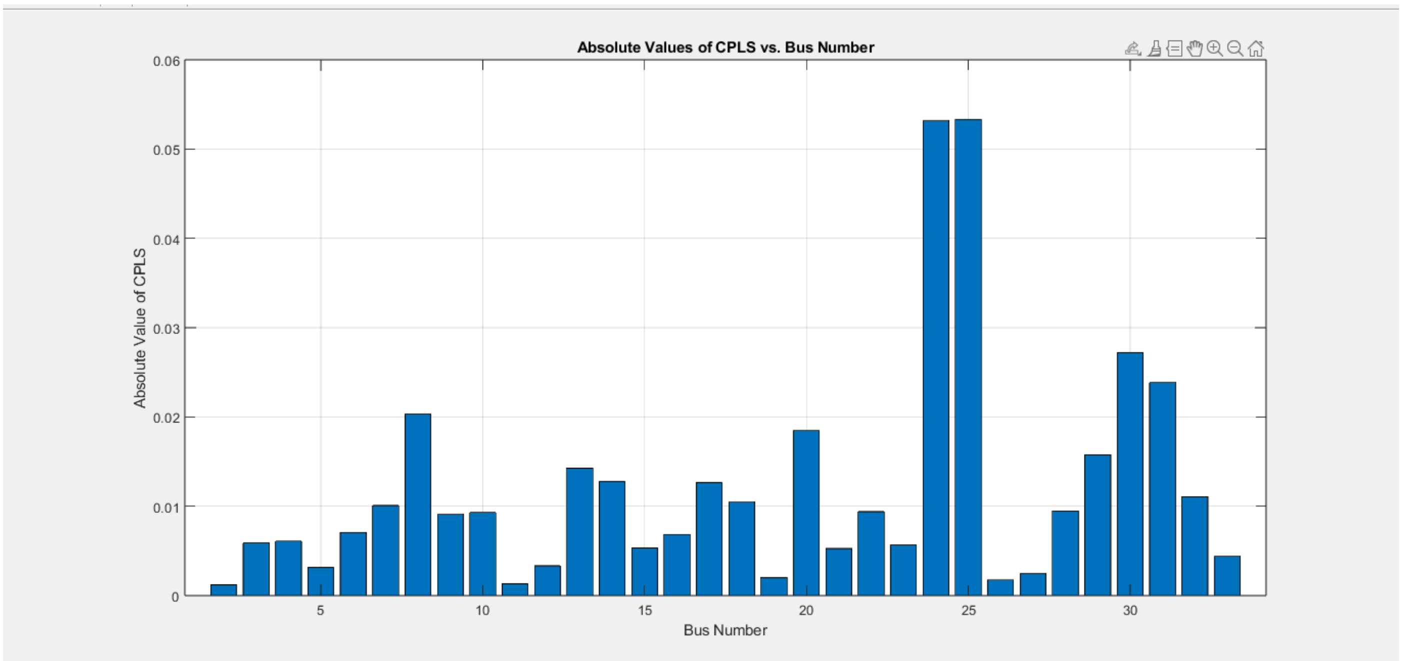
## For 33 bus system




## For 69 bus system



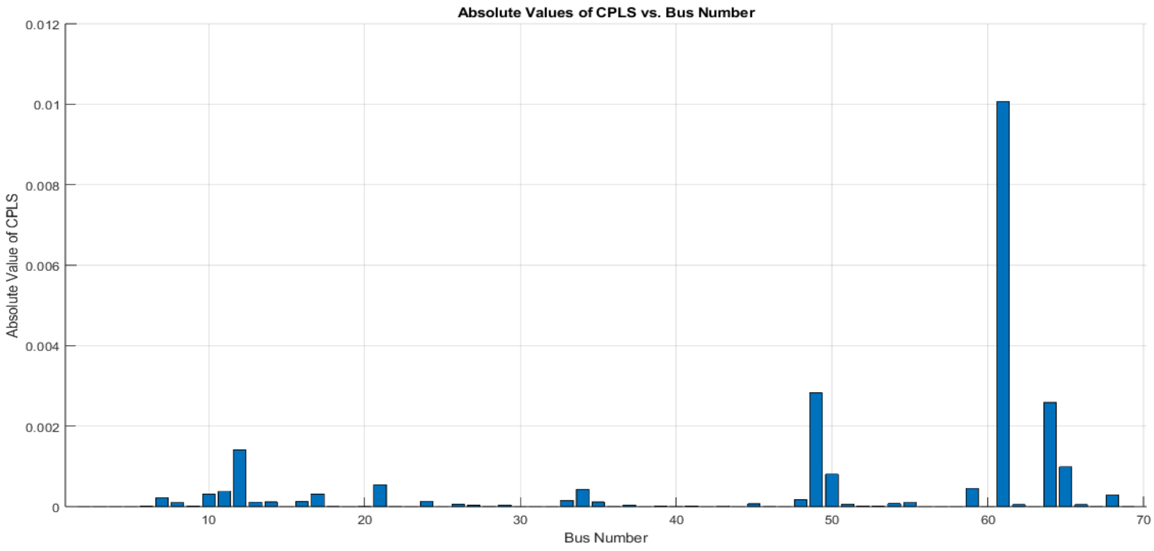
# Cpls for 33 Bus System:




# Selected Buses for 33 Bus System:

```
selected_buses =

    25
    24
    30
    31
     8
    20
    29
    13
    14
    17
    32
    18
     7
    28
    22
    10
```

Cpls for 69 Bus System:




Selected Buses for 69 Bus System :

```
selected_buses =

    61
    49
    64
    12
    65
    50
    21
    59
    34
    11
    17
    10
    68
     7
    48
    33
    24
    16
    14
    35
    13
    55
     8
    54
    45
    26
    51
    62
    66
    29
    37
    27
     9
    52
```

Final convergence : for maximum power loss reduction

Parameters are Bus no, dg size, objective function, agent no

```
alpha =

    7.0000
    0.0229
    0.0013
   16.0000


beta =

    7.0000
    0.0229
    0.0013
   24.0000


delta =

    7.0000
    0.0229
    0.0013
   49.0000
```

for 33 Bus System

```
alpha =

   61.0000
    0.0188
    0.0009
  114.0000


beta =

   61.0000
    0.0188
    0.0009
   58.0000


delta =

   61.0000
    0.0188
    0.0009
  117.0000
```

for 69 Bus System

**Selected Bus: 7 61**

For R1= 0.5, R2 = 0.5, R3 = 0

```
initial_loss =

    0.0024


final_loss =

    0.0016
```

Total power loss for 33 Bus System

```
initial_loss =

    0.0025


final_loss =

    0.0014
```

Total power loss for 69 Bus System

For R1= 0.8, R2 = 0.1, R3 = 0.1

```
initial_loss =

    0.0024


final_loss =

    0.0016
```

Total power loss for 33 Bus System

```
initial_loss =

    0.0025


final_loss =

    0.0011
```

Total power loss for 69 Bus System

For R1= 0.33 R2 = 0.33, R3 = 0.33

```
initial_loss =

    0.0024


final_loss =

    0.0022
```

Total power loss for 33 Bus System

```
initial_loss =

    0.0025


final_loss =

    0.0014
```

Total power loss for 69 Bus System

For R1= 1, R2 = 0, R3 = 0

```
initial_loss =

    0.0024


final_loss =

    0.0013
```

Total power loss for 33 Bus System

```
initial_loss =

    0.0025


final_loss =

   9.2534e-04
```

Total power loss for 69 Bus System

Maximum Power loss reduction for 33 and 69 bus system are 45.83% and 62.98% respectively.

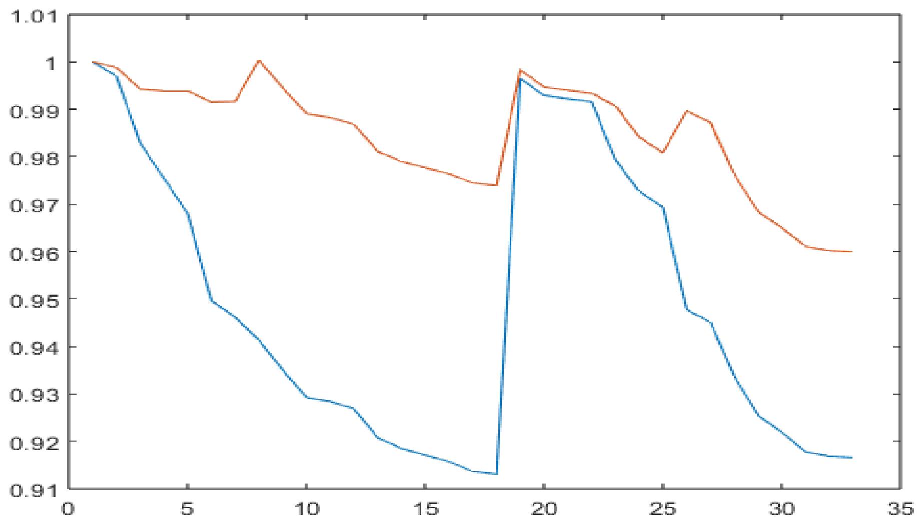
b) **PQ-type DG:**

c) Voltage Profile :

Blue : Without DG

Red : Improved

**For 33 bus system**




```
alpha =

    7.0000
    0.0250
    0.0300
    0.0121
   12.0000


beta =

    7.0000
    0.0250
    0.0300
    0.0121
   30.0000


delta =

    7.0000
    0.0250
    0.0300
    0.0121
   60.0000
```

for 33 Bus System

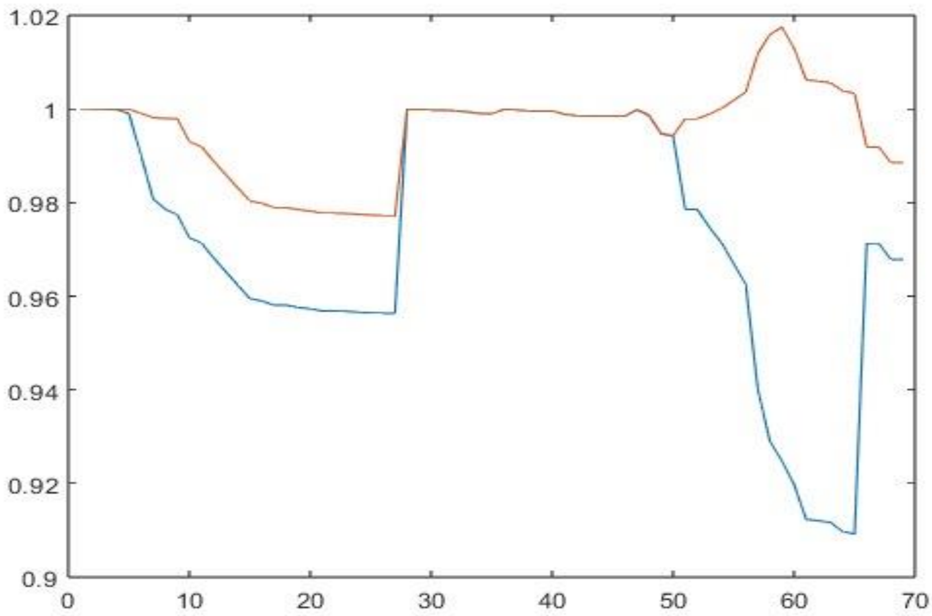
```
initial_loss =

    0.0024


final_loss =

    0.0012
```

Total power loss for 33 Bus System

```
initial_loss =

    0.0024


final_loss =

   8.2279e-04
```

Best Case R1=1, R2=0, R3=0, Total power loss for 33 Bus System

# For 69 bus system



```
alpha =

   59.0000
    0.0217
    0.0228
    0.0080
   22.0000


beta =

   59.0000
    0.0217
    0.0228
    0.0080
   55.0000


delta =

   59.0000
    0.0217
    0.0222
    0.0080
   56.0000
```

for 69 Bus System

```
initial_loss =

    0.0025


final_loss =

   7.5506e-04
```

Total power loss for 69Bus System

```
initial_loss =

    0.0025


final_loss =

   3.1656e-04
```

Best Case R1=1, R2=0, R3=0, Total power loss for 33 Bus System

**Final Results:**

## a) P-type DG

- For R1=1, R2 = 0, R3 = 0

| | 33 Bus System | 69 Bus System |
|---|---|---|
| Selected Bus for Maximum Loss Reduction | 7 | 61 |
| DG Size | 2.29 MW | 1.88 MW |
| Power Loss Reduction | 45.83 % | 62.98 % |

## b) PQ type DG

- For R1=1/3, R2 = 1/3, R3 = 1/3

| | 33 Bus System | 69 Bus System |
|---|---|---|
| Selected Bus for Maximum Loss Reduction | 7 | 59 |
| DG Size | 2.5MW , 3MVAr | 2.17MW , 2.28MVAr |
| Power Loss Reduction | 50 % | 70.12% |

- For R1=1, R2 =0 , R3 = 0 (For Max Power loss Reduction)

| | 33 Bus System | 69 Bus System |
|---|---|---|
| Selected Bus for Maximum Loss Reduction | 7 | 59 |
| Power Loss Reduction | 65.7 % | 87.34 % |

# Results from Research Paper:

- 33 Bus System:

| | | Base Case | Case 1 (Q-type) | Case 2 (P-type) | Case 3 (PQ-type) |
|---|---|---|---|---|---|
| DG Location | | - | 8 | 6 | 6 |
| DG Size | kW | - | - | 2590.2 | 2559.7 |
| | kVAR | - | 1500 | - | 1761.9 |
| PL (kW) | | 210.99 | 120.383 | 111.027 | 67.868 |
| QL (kVAR) | | 143.03 | 82.549 | 81.682 | 54.834 |
| Min bus voltage No. | | 18 | 33 | 18 | 18 |
| Vmin (p.u) | | 0.90378 | 0.93907 | 0.94237 | 0.95837 |
| VSI | | 25.5 | 27.9 | 28.5 | 29.8 |
| VD | | 1.80 | 1.07 | 0.92 | 0.56 |
| LR % | | 0.00 | 42.9 | 47.3 | 67.8 |

- 69 Bus System:

TABLE II
DG ALLOCATION IN 69-BUS SYSTEM AT DIFFERENT CASE STUDIES

| | | Base Case | Case 1 (Q-type) | Case 2 (P-type) | Case 3 (PQ-type) |
|---|---|---|---|---|---|
| DG Location | | - | 61 | 61 | 61 |
| 2*DG Size | kW | - | - | 1872.7 | 1828.5 |
| | kVAR | - | 1330 | - | 1300.6 |
| PL (kW) | | 224.99 | 152.041 | 83.222 | 23.168 |
| QL (kVAR) | | 102.197 | 70.535 | 40.568 | 14.410 |
| Min bus voltage No. | | 65 | 65 | 27 | 27 |
| Vmin (p.u) | | 0.9091 | 0.9307 | 0.96829 | 0.97247 |
| VSI | | 61.21 | 62.34 | 64.62 | 65.72 |
| VD | | 1.83 | 1.5 | 0.87 | 0.58 |
| LR % | | 0.00 | 32.4 | 63 | 89.7 |

## Comparison between Research paper and our results:

In our study, employing P-type Distributed Generators (DGs) within a 33-bus system yielded a result of 45.83% at the 7th bus. However, comparative analysis with existing research indicates a slight deviation, as prior literature reports a figure of 47.3% at the 6th bus. In our study, employing PQ-type Distributed Generators (DGs) within a 33-bus system yielded a result of 65.7% at the 7th bus. However, comparative analysis with existing research indicates a slight deviation, as prior literature reports a figure of 67.8% at the 6th bus. Similarly, within a 69-bus system, our investigation identified an optimal placement of 87.34% at the 59 bus, whereas previous research records a marginally higher value of 89.7% at the 61 bus. These disparities underscore the nuances inherent in DG placement optimization and highlight the importance of contextual factors and algorithmic intricacies in achieving optimal solutions.

## Conclusion:

In conclusion, our study aimed to enhance the efficiency of locating Distributed Generators (DGs) within power systems by employing a hybrid optimization approach. Our methodology involved a twofold strategy, integrating both analytical and heuristic techniques. Beginning with data extraction from the IEEE sample dataset, we utilized the Forward-Backward Sweep method (FBS) and its variant, the CPLS-CPLSV, to focus our analysis on the top 50% of buses exhibiting significant FBS values.

Subsequently, we formulated an objective function and applied the Grey Wolf Optimization (GWO) algorithm, along with its improved version, to iteratively optimize the placement of DGs within the power network. Through multiple iterations, we identified the optimal bus locations for two distinct systems: For PQ type DG : **bus 7 for the 33-bus system and bus 61 for the 69-bus system.** For PQ type DG : **bus 7 for the 33-bus system and bus 59 for the 69-bus system.**

Our findings underscore the efficacy of the hybrid approach in pinpointing the most suitable buses for DG integration, thereby enhancing system reliability and minimizing power losses.

## Future Scope:

In the upcoming semester, our focus will shift towards advancing our project by exploring the placement of multiple Distributed Generators (DGs) within the bus system. This expansion into multi-DG placement holds promise for enhancing system resilience and optimizing power distribution further. We aim to incorporate improvements tailored for renewable DG sources, where supplied power is a function of time, thus accommodating the variability inherent in renewable energy generation.

By implementing sophisticated optimization algorithms and leveraging comprehensive datasets, we aim to develop a robust framework capable of effectively integrating multiple DGs across various bus configurations, thereby addressing the evolving needs of modern power networks.