Multi-Objective Optimal Power Flow on the IEEE 14-Bus Test System

-Team Members:

TANIYA DAS	(23IM10039)
SOURENDRA NANDI	(23IM10045)
PRATYUSH CHATTERJEE	(23IM30017)
RAGHUVEER V	(23IM10050)
TANUSH B AGARWAL	(23IM10040)

Abstract

This project presents a comprehensive approach to solving the Multi-Objective Optimal Power Flow (MO-OPF) problem on the IEEE 14-bus test system. The optimization framework targets two key objectives: minimizing the total generation cost and enhancing the voltage profile across the network. The generation cost is modeled as a quadratic function of active power, while the voltage profile objective minimizes deviations from the nominal voltage of 1.0 per unit. A weighted sum method is used to combine both objectives into a single formulation, allowing systematic exploration of trade-offs between them.

To enhance flexibility and avoid reliance on arbitrarily fixed weight values, Pareto front analysis is additionally employed across a range of weight combinations. This allows for a more informed and structured selection of operating points that reflect a logical balance between economic efficiency and voltage stability. The approach offers a practical and adaptable framework for real-world power system optimization.

1 Introduction

Power grids are very large, complicated networks designed to deliver electricity both economically and with high reliability. As pressures accumulate—more stringent emission regulations, increased integration of renewables, and higher reliability mandates—grid managers are increasingly reaching for Optimal Power Flow (OPF) techniques.

But modern power grids demand better than cost minimization. The operators must also ensure voltage stability in order to ensure system reliability and power quality. This dual challenge has necessitated Multi-Objective OPF (MO-OPF) methodologies that balance trade-offs between conflicting objectives—like cost minimization versus voltage profile improvement.

This project is addressing two main aims:

- Minimizing overall generation cost (typically expressed as a quadratic function of generator output)
- Improving voltage quality by minimizing deviations from the ideal 1.0 per-unit voltage value

A commonly employed strategy is the weighted sum approach, which pools objectives with adjustable weights. Trade-offs may be manipulated by varying these weights by operators. It is not a straightforward activity to pick the "optimal" weights, though.

Pareto methods overcome this through the provision of a set of optimal trade-offs, referred to as the Pareto front, without predefining weights. Decision-makers may select the ideal solution based on system conditions and priorities thereafter.

This study makes use of the IEEE 14-bus test system to explain the MO-OPF approach. The actual generator mix, lines, and loads of the system are taken as a true testing environment. We make use of both weighted sum and Pareto approaches for demonstrating how efficiently operators can solve the cost–voltage trade-off problem. The structure also remains open for future objectives such as the minimization of emission or loss.

2 Literature Review

2.1 Review 1:

Optimal Power Flow (OPF) is a fundamental framework for ensuring power networks operate efficiently and safely. Traditionally, OPF was employed to reduce generation cost under voltage constraints, while recent formulations include objectives like voltage stability, power loss reduction, and emission minimization. Early OPF softwares utilized traditional optimization techniques like linear programming and quadratic programming and Newton-type approaches. Efficient for convex, smooth problems, these struggled with real-world systems not being linear and multimodal, opening the way for heuristic and metaheuristic approaches. They are more versatile but at increased computation and reduced transparency.

Among classical techniques, the weighted sum approach is common in multi-objective OPF (MO-OPF). By summing different objectives into a single function, it allows operators to alter the relative importance given to any goal through weight parameters—helpful in balancing goals like cost reduction and improvement in voltage profile.

In general, MO-OPF is interested in minimizing generation cost (expressed as a quadratic function) and maintaining voltage quality (by minimizing deviation from 1.0 p.u.). The weighted-sum method offers an operational means for investigating the trade-off, although the choice of weights must be determined in terms of system priorities.

Although Pareto-based methods portray the trade-off space in an more explicit manner, the weighted-sum method wins acceptance because it is simple, efficient, and could be readily employed as the starting point for advanced or data-intensive methodologies.

2.2 Review 2:

With growing uncertainties caused by renewable integration, market volatility, and climate change, OPF techniques have evolved towards stochastic, robust, and distributionally robust models. While these techniques enhance risk management, they increase complexity as well.

Deterministic MO-OPF remains applicable when system parameters are well established and simplicity is sought. Of them, the weighted-sum technique is widely used for its ability to aggregate multiple objectives into one function, allowing practical trade-off analysis.

3.1 Notations and Model Parameters

f : Total objective function

w_1, w_2 : Weights for cost and voltage deviation

N_g : Number of generators

N_b : Number of buses

a_i, b_i, c_i : Cost coefficients for generator i

V_{ref} : Reference voltage magnitude (typically 1.0 p.u.)

P_{Li} : Real power load at bus i

Q_{Li} : Reactive power load at bus i

G_{ij} : Real part of admittance between bus i and j

B_{ij} : Imaginary part of admittance between bus i and j

S_{ij}^{max} : Maximum apparent power flow limit on line $i \rightarrow j$

3.2 Decision Variables

P_{Gi} : Real power generated at bus i (MW)

Q_{Gi} : Reactive power generated at bus i(MVAr)

V_i : Voltage magnitude at bus i (p.u.)

θ_i : Voltage angle at bus i (radians)

3.3 Multi-Objective Function

We minimize the objective function f to achieve a trade-off between reducing generation cost (first term) and improving voltage profile by minimizing deviations from reference voltages (second term).

$$f = w_1 \cdot \sum_{i=1}^{N_g} \left(a_i P_{Gi}^2 + b_i P_{Gi} + c_i \right) + w_2 \cdot \sum_{j=1}^{N_b} \left(V_j - V_{ref} \right)^2$$

3.4 Constraints

Equality Constraints (Power Flow Equations)

1. Active Power Balance: (for all i in Nb)

$$P_{G_i} - P_{L_i} = V_i \sum_{j=1}^{N_b} V_j \left[G_{ij}\cos(\theta_i - \theta_j) + B_{ij}\sin(\theta_i - \theta_j) \right]$$

2. Reactive Power Balance: (for all i in Nb)

$$Q_{G_i} - Q_{L_i} = V_i \sum_{j=1}^{N_b} V_j \left[G_{ij}\sin(\theta_i - \theta_j) - B_{ij}\cos(\theta_i - \theta_j) \right]$$

Inequality Constraints

1. Generator Operating Limits: (for all i in Nb)

$$P_{G_i}^{min} \leq P_{G_i} \leq P_{G_i}^{max}$$

$$Q_{G_i}^{min} \leq Q_{G_i} \leq Q_{G_i}^{max}$$

2. Voltage Magnitude Limits: (for all i in Nb)

$$V_i^{min} \leq V_i \leq V_i^{max} \quad (\text{usually } 0.95 \leq V_i \leq 1.05 \text{ p.u.})$$

3. Line Flow Limits: (for all i, j in Nb)

$$S_{ij} = \sqrt{P_{ij}^2 + Q_{ij}^2} \leq S_{ij}^{max}$$

4. Slack Bus Constraint:

$$V_1 = 1.0 \text{ p.u.} \quad \theta_1 = 0 \text{ rad}$$

4 Methodology

4.1 Method 1:

Optimal power flow (OPF) analysis was performed on the base-case IEEE 14-bus system of MATPOWER's case14 dataset in MATLAB. The generator cost matrix (gencost) was verified and updated to ensure quadratic cost coefficients were defined for all five generators. Major system data—base MVA, bus admittance matrix (Ybus), generator limits, voltage limits, and loads—were extracted. Here, the script also stored the original bus and branch data into two Excel files (bus_data.xlsx, branch_data.xlsx)(in Appendix) with load, voltage, and line parameter values for note-taking. The Ybus matrix was divided into real (G) and imaginary (B) components for use in power balance equations, and generator cost coefficients (a, b, c) were utilized to define the cost objective.

Optimization parameters were **Bus Voltage magnitudes**, **Voltage Angles**, and **Real and Reactive power outputs of generators (Pg and Qg).** Initial guesses were specified within feasible limits.

A weighted objective function was formulated based on total generation cost and voltage deviation from 1.0 p.u. and their respective weights as 0.7 and 0.3. Constraints were set to ensure power balance at all buses, apply generator and voltage limits, fix the slack bus angle to 0°, and restrict line flows to below 100 MVA.

The nonlinear optimization problem was solved by **MATLAB's fmincon** using the **Interior-point algorithm** and high iteration and tight tolerance parameters to ensure accuracy. Finally, **voltage and angle plots** were generated in order to plot the optimal solution.

4.2 Method 2:

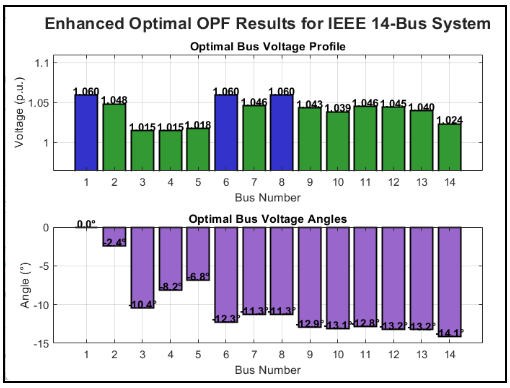
As a continuation of the previous multi-objective OPF solution, a **Pareto front-based approach** was employed to examine generation cost versus voltage deviation trade-offs in a systematic manner. Instead of employing fixed weights, the epsilon-constraint method was used to generate a set of Pareto-optimal solutions by adjusting the permissible voltage deviation over a pre-specified range. For each solution point, the cost was minimized with an upper limit on voltage deviation. A Pareto front was constructed using these solutions for the range of cost-optimal to voltage-optimal operation.

Gradients between nearby Pareto points were utilized to compute equivalent weight pairs (w_1 for cost, w_2 for voltage deviation) per solution. Weights were normalized and expressed to demonstrate balanced trade-offs, dominating objectives, and ideal configurations to aid in making choices. The approach facilitated improved informed decisions among OPF solutions based on system priorities rather than predetermined fixed weightings.

5 Results

5.1 Results for Method 1:

The adjoining graph illustrates a well-controlled voltage and angle profile achieved under the weighted sum optimization for the IEEE 14-bus system. As shown in the bar plots, all bus voltages lie within acceptable operational limits, with generator buses (1, 6, and 8) maintaining the highest



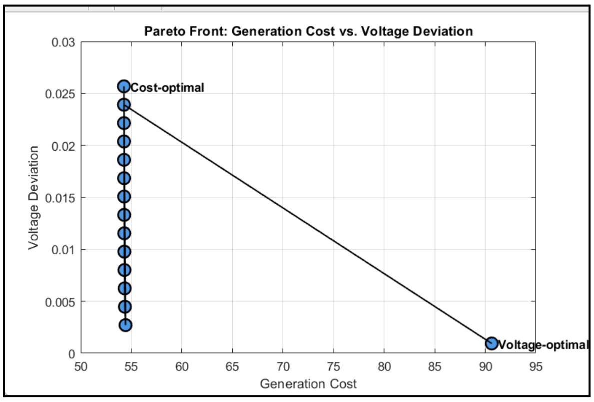
voltage level of 1.060 p.u. and the lowest voltages, approximately 1.015 p.u., observed at Buses 3 and 4. The voltage angles display a smooth and continuous decline from 0° at the slack bus (Bus 1) to −14.1° at Bus 14, indicating stable and consistent directional power flow across the network.

These trends are also confirmed by the numerical results, which show the effectiveness of the optimization in fulfilling both goals. The **cost of generation** was reduced to a minimum of **54.2712**, showcasing economic effectiveness, and the **voltage profile metric** reduced to **0.025657**, signifying low deviation from nominal voltage. Only Buses 1 and 2 produced real power with generations of 138.07 MW and 130.74 MW, respectively. Reactive support was given by Buses 3, 6, and 8. Overall, the outcomes verify that the weighted sum approach effectively balances cost reduction with voltage stability throughout the network.

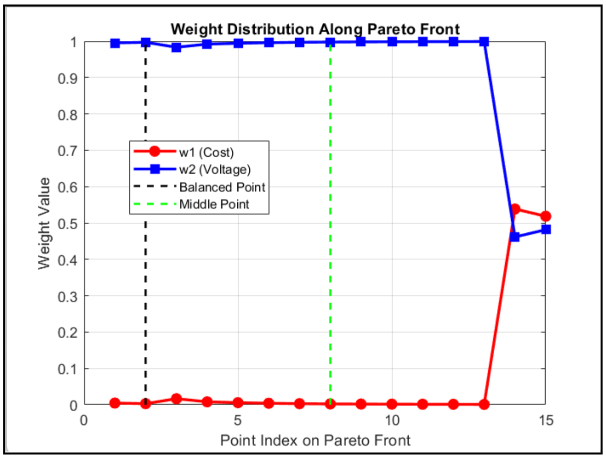
```
Optimal Bus Voltages and Angles:
Bus  1: V = 1.0600 p.u., theta = 0.0000 rad (0.00°)
Bus  2: V = 1.0485 p.u., theta = -0.0423 rad (-2.42°)
Bus  3: V = 1.0148 p.u., theta = -0.1822 rad (-10.44°)
Bus  4: V = 1.0153 p.u., theta = -0.1428 rad (-8.18°)
Bus  5: V = 1.0177 p.u., theta = -0.1193 rad (-6.84°)
Bus  6: V = 1.0600 p.u., theta = -0.2149 rad (-12.31°)
Bus  7: V = 1.0463 p.u., theta = -0.1971 rad (-11.29°)
Bus  8: V = 1.0600 p.u., theta = -0.1971 rad (-11.29°)
Bus  9: V = 1.0432 p.u., theta = -0.2255 rad (-12.92°)
Bus 10: V = 1.0387 p.u., theta = -0.2287 rad (-13.10°)
Bus 11: V = 1.0457 p.u., theta = -0.2241 rad (-12.84°)
Bus 12: V = 1.0448 p.u., theta = -0.2300 rad (-13.18°)
Bus 13: V = 1.0398 p.u., theta = -0.2312 rad (-13.25°)
Bus 14: V = 1.0235 p.u., theta = -0.2459 rad (-14.09°)
Optimal Generator Outputs:
Generator at Bus  1: Pg = 1.3807 p.u. (138.07 MW), Qg = 0.0000 p.u. (0.00 MVAr)
Generator at Bus  2: Pg = 1.3074 p.u. (130.74 MW), Qg = 0.2223 p.u. (22.23 MVAr)
Generator at Bus  3: Pg = 0.0000 p.u. (0.00 MW), Qg = 0.2974 p.u. (29.74 MVAr)
Generator at Bus  6: Pg = 0.0000 p.u. (0.00 MW), Qg = 0.1054 p.u. (10.54 MVAr)
Generator at Bus  8: Pg = 0.0000 p.u. (0.00 MW), Qg = 0.0826 p.u. (8.26 MVAr)
Total Generation Cost: 54.2712
Voltage Profile Metric: 0.025657
```

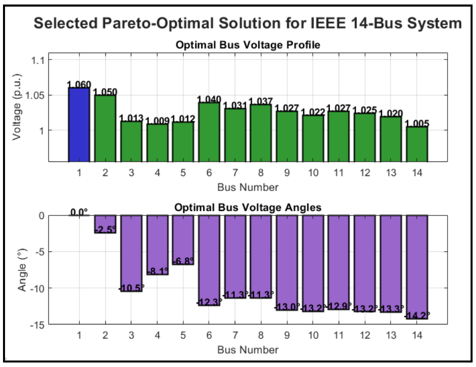
5.2 Results for Method 2:

The Pareto front achieved by the epsilon-constraint method shows the trade-off relationship between voltage deviation and generation cost. The curve shows that reducing generation cost leads to larger voltage deviations, and the reverse. This reverse relationship illustrates the nature of multi-objective optimization—where the improvement in one objective normally sacrifices another. The smooth continuity of points along the front reveals a well-behaved optimization process and a well-balanced range of feasible solutions.



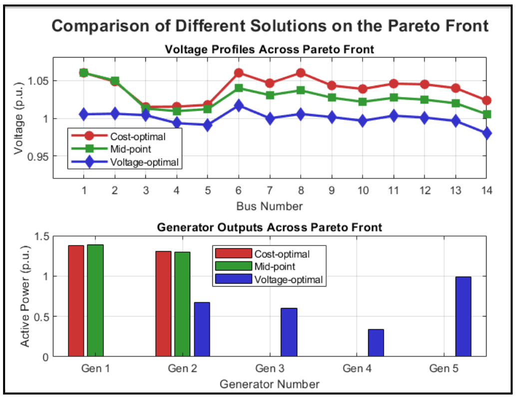
In the adjoining figure, discrete Pareto-optimal solutions are plotted for selected values of the epsilon constraint. Each point represents an optimal solution obtained for a fixed upper bound on voltage deviation. The graph illustrates how the generation cost decreases gradually as the permissible deviation increases. This confirms that the cost function responds effectively to variations in the constraint and enables a range of optimal trade-off solutions to be explored based on operational priorities.





The bar chart shows how generation cost decreases with increasing epsilon values. As the voltage deviation constraint is relaxed, the cost drops steadily, highlighting the benefit of allowing slight flexibility. This supports earlier findings and offers a clearer comparison across discrete scenarios for informed cost optimization.

This figure corroborates the cost analysis by showing the respective voltage deviation profile for each epsilon value. An increasing deviation trend is apparent, which exactly replicates the cost savings seen earlier. This verifies the expected trade-off pattern, as more relaxed deviation limits allow the system to be more cost-effective but with poorer voltage quality.




```
Detailed Results for Pareto Solution:
Generation Cost: 54.2961, Voltage Deviation: 0.013311

Optimal Bus Voltages and Angles:
Bus  1: V = 1.0600 p.u., theta = 0.0000 rad (0.00°)
Bus  2: V = 1.0498 p.u., theta = -0.0433 rad (-2.48°)
Bus  3: V = 1.0128 p.u., theta = -0.1829 rad (-10.48°)
Bus  4: V = 1.0094 p.u., theta = -0.1421 rad (-8.14°)
Bus  5: V = 1.0120 p.u., theta = -0.1184 rad (-6.78°)
Bus  6: V = 1.0397 p.u., theta = -0.2155 rad (-12.35°)
Bus  7: V = 1.0306 p.u., theta = -0.1979 rad (-11.34°)
Bus  8: V = 1.0368 p.u., theta = -0.1979 rad (-11.34°)
Bus  9: V = 1.0271 p.u., theta = -0.2275 rad (-13.03°)
Bus 10: V = 1.0217 p.u., theta = -0.2306 rad (-13.21°)
Bus 11: V = 1.0271 p.u., theta = -0.2255 rad (-12.92°)
Bus 12: V = 1.0246 p.u., theta = -0.2312 rad (-13.25°)
Bus 13: V = 1.0198 p.u., theta = -0.2326 rad (-13.33°)
Bus 14: V = 1.0054 p.u., theta = -0.2482 rad (-14.22°)
```

These graphs are confirmed by the numerical results, which show the effectiveness of using the Pareto front analysis for optimization in fulfilling both goals. The **cost of**

```
Optimal Generator Outputs:
Generator at Bus  1: Pg = 1.3900 p.u. (139.00 MW), Qg = 0.0000 p.u. (0.00 MVAr)
Generator at Bus  2: Pg = 1.2995 p.u. (129.95 MW), Qg = 0.3493 p.u. (34.93 MVAr)
Generator at Bus  3: Pg = 0.0000 p.u. (0.00 MW), Qg = 0.3039 p.u. (30.39 MVAr)
Generator at Bus  6: Pg = 0.0000 p.u. (0.00 MW), Qg = 0.0349 p.u. (3.49 MVAr)
Generator at Bus  8: Pg = 0.0000 p.u. (0.00 MW), Qg = 0.0368 p.u. (3.68 MVAr)
```

generation was reduced to a minimum of **54.2961**, and the **voltage profile metric** reduced to **0.013311**, signifying low deviation from nominal voltage. Buses 1 and 2 produced real power with generations of 139.00 MW and 129.95 MW, respectively. Reactive support was given by Buses 3, 6, and 8. Overall, the outcomes show that the Pareto front analysis is a logical and effective way to deal with real-life multi-objective functions.

6 Conclusion

This project achieves a technique of solving the Multi-Objective Optimal Power Flow (MO-OPF) problem using the IEEE 14-bus test system. By applying dual goals of reducing total generation cost and enhancing voltage profile, the research gives a practicable framework for solving economic and stability concerns in power system operation.

Though the weighted sum approach proved effective to combine the cost and voltage objectives into one function , Pareto front analysis was employed to generate a collection of optimal, non-dominated solutions that offer a broader perspective of the balance of cost and voltage. The approach is devoid of the limitations of relying on a single set of pre-specified weights and allows operators to select best-suited solutions according to available system conditions.

The application of both techniques in the MO-OPF framework equips power system operators with enhanced decision-making capacities to deal with the advanced and dynamic requirements of modern power systems, enabling wiser and robust operating strategies.

7 References

Reference 1: *Ebeed, M., Kamel, S., & Jurado, F. (2018). Optimal Power Flow Using Recent Optimization Techniques. In Classical and Recent Aspects of Power System Optimization (Chapter 7, pp. 157-183). Academic Press.*

Reference 2: *Roald, L. A., Pozo, D., Papavassiliou, A., Molzahn, D. K., Kazempour, J., & Conejo, A. (2023). Power systems optimization under uncertainty: A review of methods and applications. Electric Power Systems Research, 214, 108725.*
https://www.elsevier.com/locate/epsr

8 Appendix

Method 1:

```matlab
function analyze_opf_case14()
    % Load MATPOWER's 14-bus case and define constants
    mpc = case14;
    define_constants;

    % Ensure generator cost data exists (5 generators for case14)
    if size(mpc.gencost, 2) < 7
        mpc.gencost = [
            2, 0, 0, 2, 0.02, 10, 100;
            2, 0, 0, 2, 0.04, 8, 80;
            2, 0, 0, 2, 0.03, 12, 120;
            2, 0, 0, 2, 0.05, 15, 150;
            2, 0, 0, 2, 0.025, 9, 90;
        ];
    end
    % Set branch limits (RATE_A)
    mpc.branch(:, RATE_A) = 100;  % 100 MVA limit

    %% Extract System Parameters
    baseMVA = mpc.baseMVA;
    nb = size(mpc.bus, 1);      % Number of buses (14 for case14)
    ng = size(mpc.gen, 1);      % Number of generators (5 for case14)
    nl = size(mpc.branch, 1);   % Number of branches
    slack_bus = find(mpc.bus(:, BUS_TYPE) == 3);

    % Admittance matrix
    [Ybus, ~, ~] = makeYbus(baseMVA, mpc.bus, mpc.branch);
    G = real(Ybus);
    B = imag(Ybus);
    % Generator data
    Pmin = mpc.gen(:, PMIN) / baseMVA;
    Pmax = mpc.gen(:, PMAX) / baseMVA;
    Qmin = mpc.gen(:, QMIN) / baseMVA;
    Qmax = mpc.gen(:, QMAX) / baseMVA;
    if size(mpc.gencost, 2) >= 7
        a = mpc.gencost(:, 5);
        b = mpc.gencost(:, 6);
        c = mpc.gencost(:, 7);
    else
        a = 0.01 * ones(ng, 1);
        b = 10 * ones(ng, 1);
        c = 100 * ones(ng, 1);
    end
    Vmin = mpc.bus(:, VMIN);
    Vmax = mpc.bus(:, VMAX);
    Pd = mpc.bus(:, PD) / baseMVA;
    Qd = mpc.bus(:, QD) / baseMVA;
    %% Optimization Setup
    % Weighting factors for multi-objective (generation cost and voltage profile)
    w1 = 0.7;  % Weight on generation cost
    w2 = 0.3;  % Weight on voltage deviations
    Vref = 1.0;
    % Initial guess for decision variables: [V; theta; Pg; Qg]
    V0 = ones(nb, 1);
    theta0 = zeros(nb, 1);
    Pg0 = zeros(ng, 1);
    Qg0 = zeros(ng, 1);
    % Map generator bus indices & set mid-range generation values
    gen_buses = mpc.gen(:, 1);
    for i = 1:ng
        Pg0(i) = (Pmax(i) + Pmin(i)) / 2;
        Qg0(i) = (Qmax(i) + Qmin(i)) / 2;
    end
    x0 = [V0; theta0; Pg0; Qg0];
    % Define variable bounds
    lb_V = Vmin;
    ub_V = Vmax;
    lb_theta = -pi * ones(nb, 1);
    ub_theta = pi * ones(nb, 1);
    lb_theta(slack_bus) = 0;
    ub_theta(slack_bus) = 0;
    lb_Pg = Pmin;
    ub_Pg = Pmax;
    lb_Qg = Qmin;
    ub_Qg = Qmax;
    lb = [lb_V; lb_theta; lb_Pg; lb_Qg];
    ub = [ub_V; ub_theta; ub_Pg; ub_Qg];
    %% Define Objective and Constraint Functions
    objfun = @(x) opf_objective(x, nb, ng, gen_buses, a, b, c, w1, w2, Vref);
    nonlcon = @(x) opf_constraints(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, mpc.branch(:, RATE_
    %% Optimization Options and Solve with fmincon
    options = optimoptions('fmincon', 'Algorithm', 'interior-point', ...
        'Display', 'iter', 'MaxFunctionEvaluations', 10000, ...
        'MaxIterations', 1000, 'OptimalityTolerance', 1e-6, 'ConstraintTolerance', 1e-6);
    [x_opt, fval, exitflag, output] = fmincon(objfun, x0, [], [], [], [], lb, ub, nonlcon, options);
    %% Extract Results
    V_opt = x_opt(1:nb);
    theta_opt = x_opt(nb+1:2*nb);
    Pg_opt = x_opt(2*nb+1:2*nb+ng);
    Qg_opt = x_opt(2*nb+ng+1:end);
    %% Display Numerical Results
    disp('Optimal Bus Voltages and Angles:');
    for i = 1:nb
        fprintf('Bus %2d: V = %.4f p.u., theta = %.4f rad (%.2f°)\n', ...
            i, V_opt(i), theta_opt(i), theta_opt(i)*180/pi);
    end
    disp('Optimal Generator Outputs:');
    for i = 1:ng
```

```
        fprintf('Generator at Bus %2d: Pg = %.4f p.u. (%.2f MW), Qg = %.4f p.u. (%.2f MVAr)\n', ...
            gen_buses(i), Pg_opt(i), Pg_opt(i)*baseMVA, Qg_opt(i), Qg_opt(i)*baseMVA);
    end
    total_cost = sum(a .* Pg_opt.^2 + b .* Pg_opt + c);
    fprintf('Total Generation Cost: %.4f\n', total_cost);
    voltage_metric = sum((V_opt - Vref).^2);
    fprintf('Voltage Profile Metric: %.6f\n', voltage_metric);
    % losses = calculateLosses(V_opt, theta_opt, mpc.branch, baseMVA);
    % fprintf('Total System Losses: %.4f MW\n', losses);
    %% Enhanced Visual Outputs
    figure('Name', 'Optimal OPF Results - IEEE 14-Bus', 'Color', [1 1 1]);
    t = tiledlayout(2, 1, 'Padding', 'compact', 'TileSpacing', 'compact');
    % Voltage Profile Plot
    nexttile;
    hBarVolt = bar(V_opt, 'FaceColor', 'flat', 'EdgeColor', 'k', 'LineWidth', 1.2);
    % Apply gradient coloring based on voltage levels
    for k = 1:length(V_opt)
        if V_opt(k) < 0.95
            hBarVolt.CData(k,:) = [0.8 0.2 0.2];  % Red tones for low voltages
        elseif V_opt(k) > 1.05
            hBarVolt.CData(k,:) = [0.2 0.2 0.8];  % Blue tones for high voltages
        else
            hBarVolt.CData(k,:) = [0.2 0.6 0.2];  % Green for acceptable voltage
        end
    end
    grid on; box on;
    xlabel('Bus Number');
    ylabel('Voltage (p.u.)');
    title('Optimal Bus Voltage Profile');
    ylim([min(V_opt)-0.05, max(V_opt)+0.05]);
    xticks(1:length(V_opt));
    % Annotate each bar with its voltage value
    for k = 1:length(V_opt)
        text(k, V_opt(k)+0.005, sprintf('%.3f', V_opt(k)), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold');
    end
    % Voltage Angle Plot
    nexttile;
    hBarAngle = bar(theta_opt*180/pi, 'FaceColor', [0.6 0.4 0.8], 'EdgeColor', 'k', 'LineWidth', 1.2);
    grid on; box on;
    xlabel('Bus Number');
    ylabel('Angle (°)');
    title('Optimal Bus Voltage Angles');
    xticks(1:length(theta_opt));
    % Annotate each angle value (in degrees)
    for k = 1:length(theta_opt)
        text(k, (theta_opt(k)*180/pi)+0.5, sprintf('%.1f°', theta_opt(k)*180/pi), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold');
    end
    % Add an overall title for the tiled layout
    title(t, 'Enhanced Optimal OPF Results for IEEE 14-Bus System', 'FontSize', 14, 'FontWeight', 'bold'
end
%% Objective Function
function f = opf_objective(x, nb, ng, gen_buses, a, b, c, w1, w2, Vref)
    % Extract voltage and generator power values
    V = x(1:nb);
    Pg = x(2*nb+1:2*nb+ng);
    % Generation cost (quadratic model)
    gen_cost = sum(a .* Pg.^2 + b .* Pg + c);
    % Voltage deviation cost
    voltage_term = sum((V - Vref).^2);
    % Weighted combined objective
    f = w1 * gen_cost + w2 * voltage_term;
end
%% Nonlinear Constraints
function [c, ceq] = opf_constraints(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, Smax)
    % Extract decision variables
    V = x(1:nb);
    theta = x(nb+1:2*nb);
    Pg = x(2*nb+1:2*nb+ng);
    Qg = x(2*nb+ng+1:2*nb+2*ng);
    % Map generator outputs to their buses
    Pg_bus = zeros(nb, 1);
    Qg_bus = zeros(nb, 1);
    for i = 1:ng
        Pg_bus(gen_buses(i)) = Pg(i);
        Qg_bus(gen_buses(i)) = Qg(i);
    end
    % Power balance equations for each bus
    Pbalance = zeros(nb, 1);
    Qbalance = zeros(nb, 1);
    for i = 1:nb
        P_inj = 0;
        Q_inj = 0;
        for j = 1:nb
            angle_diff = theta(i) - theta(j);
            P_inj = P_inj + V(i) * V(j) * (G(i,j) * cos(angle_diff) + B(i,j) * sin(angle_diff));
            Q_inj = Q_inj + V(i) * V(j) * (G(i,j) * sin(angle_diff) - B(i,j) * cos(angle_diff));
        end
        Pbalance(i) = Pg_bus(i) - Pd(i) - P_inj;
        Qbalance(i) = Qg_bus(i) - Qd(i) - Q_inj;
    end
    % Line flow constraints
    line_flow = [];
    if ~isempty(Smax) && Smax(1) ~= 999
        line_flow = computeLineFlows(V, theta, nb, nl, G, B, Smax);
    end
    c = line_flow;
    ceq = [Pbalance; Qbalance];
end
```

```matlab
%% Compute Line Flows
function c = computeLineFlows(V, theta, nb, nl, G, B, Smax)
    c = [];
    line_count = 0;
    for i = 1:nb
        for j = i+1:nb
            if abs(G(i,j)) > 1e-6 || abs(B(i,j)) > 1e-6
                line_count = line_count + 1;
                if line_count <= nl
                    angle_diff = theta(i) - theta(j);
                    Pij = V(i)^2 * G(i,j) - V(i)*V(j) * (G(i,j)*cos(angle_diff) + B(i,j)*sin(angle_diff));
                    Qij = -V(i)^2 * B(i,j) - V(i)*V(j) * (G(i,j)*sin(angle_diff) - B(i,j)*cos(angle_diff));
                    Sij = sqrt(Pij^2 + Qij^2);
                    c = [c; Sij - Smax(line_count)];
                end
            end
        end
    end
end

```

Method 2:

```matlab
function analyze_opf_case14_pareto()
    % Load MATPOWER's 14-bus case and define constants
    mpc = case14;
    define_constants;
    % Ensure generator cost data exists (5 generators for case14)
    if size(mpc.gencost, 2) < 7
        mpc.gencost = [
            2, 0, 0, 2, 0.02, 10, 100;
            2, 0, 0, 2, 0.04, 8, 80;
            2, 0, 0, 2, 0.03, 12, 120;
            2, 0, 0, 2, 0.05, 15, 150;
            2, 0, 0, 2, 0.025, 9, 90;
        ];
    end
    % Set branch limits (RATE_A)
    mpc.branch(:, RATE_A) = 100;  % 100 MVA limit
    %% Extract System Parameters
    baseMVA = mpc.baseMVA;
    nb = size(mpc.bus, 1);      % Number of buses (14 for case14)
    ng = size(mpc.gen, 1);      % Number of generators (5 for case14)
    nl = size(mpc.branch, 1);   % Number of branches
    slack_bus = find(mpc.bus(:, BUS_TYPE) == 3);
    % Admittance matrix
    [Ybus, ~, ~] = makeYbus(baseMVA, mpc.bus, mpc.branch);
    G = real(Ybus);
    B = imag(Ybus);
    % Generator data
    Pmin = mpc.gen(:, PMIN) / baseMVA;
    Pmax = mpc.gen(:, PMAX) / baseMVA;
    Qmin = mpc.gen(:, QMIN) / baseMVA;
    Qmax = mpc.gen(:, QMAX) / baseMVA;
    if size(mpc.gencost, 2) >= 7
        a = mpc.gencost(:, 5);
        b = mpc.gencost(:, 6);
        c = mpc.gencost(:, 7);
    else
        a = 0.01 * ones(ng, 1);
        b = 10 * ones(ng, 1);
        c = 100 * ones(ng, 1);
    end
    Vmin = mpc.bus(:, VMIN);
    Vmax = mpc.bus(:, VMAX);
    Pd = mpc.bus(:, PD) / baseMVA;
    Qd = mpc.bus(:, QD) / baseMVA;
    %% Optimization Setup
    Vref = 1.0;
    % Initial guess for decision variables: [V; theta; Pg; Qg]
    V0 = ones(nb, 1);
    theta0 = zeros(nb, 1);
    Pg0 = zeros(ng, 1);
    Qg0 = zeros(ng, 1);
    % Map generator bus indices & set mid-range generation values
    gen_buses = mpc.gen(:, 1);
    for i = 1:ng
        Pg0(i) = (Pmax(i) + Pmin(i)) / 2;
        Qg0(i) = (Qmax(i) + Qmin(i)) / 2;
    end
    x0 = [V0; theta0; Pg0; Qg0];
    % Define variable bounds
    lb_V = Vmin;
    ub_V = Vmax;
    lb_theta = -pi * ones(nb, 1);
    ub_theta = pi * ones(nb, 1);
    lb_theta(slack_bus) = 0;
    ub_theta(slack_bus) = 0;
    lb_Pg = Pmin;
    ub_Pg = Pmax;
    lb_Qg = Qmin;
    ub_Qg = Qmax;
    lb = [lb_V; lb_theta; lb_Pg; lb_Qg];
    ub = [ub_V; ub_theta; ub_Pg; ub_Qg];
    %% Generate Pareto Front using epsilon-constraint method
    % Define number of points on the Pareto front
    num_points = 15;
    % Find the range of each objective
    % First, optimize for cost only
    objfun_cost = @(x) opf_cost_objective(x, nb, ng, gen_buses, a, b, c);
    nonlcon = @(x) opf_constraints(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, mpc.branch(:, RATE_A)/baseMVA);
    options = optimoptions('fmincon', 'Algorithm', 'interior-point', ...
        'Display', 'iter', 'MaxFunctionEvaluations', 10000, ...
        'MaxIterations', 1000, 'OptimalityTolerance', 1e-6, 'ConstraintTolerance', 1e-6);
    [x_cost, fval_cost, ~, ~] = fmincon(objfun_cost, x0, [], [], [], [], lb, ub, nonlcon, options);
    % Extract cost-optimal results
    V_cost = x_cost(1:nb);
    voltage_dev_at_cost_opt = sum((V_cost - Vref).^2);
```

```matlab
    fprintf('Min Cost Solution: Cost = %.4f, Voltage Deviation = %.6f\n', fval_cost, voltage_dev_at_cost_opt);
    % Then, optimize for voltage profile only
    objfun_voltage = @(x) opf_voltage_objective(x, nb, Vref);
    [x_volt, fval_volt, ~, ~] = fmincon(objfun_voltage, x0, [], [], [], [], lb, ub, nonlcon, options);
    % Extract voltage-optimal results
    cost_at_volt_opt = opf_cost_objective(x_volt, nb, ng, gen_buses, a, b, c);
    fprintf('Min Voltage Deviation Solution: Cost = %.4f, Voltage Deviation = %.6f\n', cost_at_volt_opt, fval_volt);
    % Define the range of epsilon values
    if voltage_dev_at_cost_opt > fval_volt
        epsilon_values = linspace(fval_volt, voltage_dev_at_cost_opt, num_points);
    else
        epsilon_values = linspace(fval_volt, 2*voltage_dev_at_cost_opt, num_points);
    end
    % Create arrays to store Pareto front results
    pareto_cost = zeros(num_points, 1);
    pareto_voltage_dev = zeros(num_points, 1);
    pareto_solutions = cell(num_points, 1);
    % Store cost-optimal solution
    pareto_cost(1) = fval_cost;
    pareto_voltage_dev(1) = voltage_dev_at_cost_opt;
    pareto_solutions{1} = x_cost;
    % Store voltage-optimal solution
    pareto_cost(num_points) = cost_at_volt_opt;
    pareto_voltage_dev(num_points) = fval_volt;
    pareto_solutions{num_points} = x_volt;
    % Solve for Pareto-optimal solutions using epsilon constraint method
    for i = 2:num_points-1
        epsilon = epsilon_values(i);
        fprintf('\nSolving for epsilon = %.6f (%d/%d)\n', epsilon, i, num_points);
        % Create constraint function with voltage deviation limit
        nonlcon_epsilon = @(x) opf_constraints_with_epsilon(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, mpc.branch
        % Solve with cost as objective and voltage deviation as constraint
        [x_pareto, fval_pareto, exitflag, ~] = fmincon(objfun_cost, x0, [], [], [], [], lb, ub, nonlcon_epsilon, options);
        if exitflag > 0
            % Store successful solution
            pareto_cost(i) = fval_pareto;
            pareto_voltage_dev(i) = opf_voltage_objective(x_pareto, nb, Vref);
            pareto_solutions{i} = x_pareto;
            fprintf('Found solution: Cost = %.4f, Voltage Deviation = %.6f\n', pareto_cost(i), pareto_voltage_dev(i));
        else
            fprintf('No solution found for epsilon = %.6f\n', epsilon);
            % Use previous solution
            pareto_cost(i) = pareto_cost(i-1);
            pareto_voltage_dev(i) = pareto_voltage_dev(i-1);
            pareto_solutions{i} = pareto_solutions{i-1};
        end
    end
    % Filter invalid or identical solutions
    idx = 1;
    valid_cost = pareto_cost(idx);
    valid_volt = pareto_voltage_dev(idx);
    valid_solutions = {pareto_solutions{idx}};
    for i = 2:num_points
        if ~isnan(pareto_cost(i)) && ~isnan(pareto_voltage_dev(i)) && ...
           (abs(pareto_cost(i) - valid_cost(end)) > 1e-4 || abs(pareto_voltage_dev(i) - valid_volt(end)) > 1e-6)
            valid_cost = [valid_cost; pareto_cost(i)];
            valid_volt = [valid_volt; pareto_voltage_dev(i)];
            valid_solutions = [valid_solutions; pareto_solutions(i)];
        end
    end
    %% Display and Visualize Pareto Front
    % Plot Pareto front
    figure('Name', 'Pareto Front - OPF Multi-Objective', 'Color', [1 1 1]);
    scatter(valid_cost, valid_volt, 100, 'filled', 'MarkerFaceColor', [0.3 0.6 0.9], 'MarkerEdgeColor', 'k', 'LineWidth', 1.5);
    hold on;
    plot(valid_cost, valid_volt, 'k-', 'LineWidth', 1.2);
    xlabel('Generation Cost');
    ylabel('Voltage Deviation');
    title('Pareto Front: Generation Cost vs. Voltage Deviation');
    grid on; box on;
    text(valid_cost(1), valid_volt(1), '  Cost-optimal', 'FontWeight', 'bold');
    text(valid_cost(end), valid_volt(end), '  Voltage-optimal', 'FontWeight', 'bold');
    % Extract the solution for detailed analysis (mid-point of Pareto front as an example)
    mid_idx = ceil(length(valid_cost) / 2);
    x_selected = valid_solutions{mid_idx};
    % Extract values from selected solution
    V_opt = x_selected(1:nb);
    theta_opt = x_selected(nb+1:2*nb);
    Pg_opt = x_selected(2*nb+1:2*nb+ng);
    Qg_opt = x_selected(2*nb+ng+1:end);
    % Extract weights from the Pareto front
    extractWeightsFromPareto(valid_cost, valid_volt, valid_solutions);
    %% Display Numerical Results for selected solution
    fprintf('\n\nDetailed Results for Pareto Solution:\n');
    fprintf('Generation Cost: %.4f, Voltage Deviation: %.6f\n', valid_cost(mid_idx), valid_volt(mid_idx));
    fprintf('\nOptimal Bus Voltages and Angles:\n');
    for i = 1:nb
        fprintf('Bus %2d: V = %.4f p.u., theta = %.4f rad (%.2f°)\n', ...
            i, V_opt(i), theta_opt(i), theta_opt(i)*180/pi);
    end
    fprintf('\nOptimal Generator Outputs:\n');
    for i = 1:ng
        fprintf('Generator at Bus %2d: Pg = %.4f p.u. (%.2f MW), Qg = %.4f p.u. (%.2f MVAr)\n', ...
            gen_buses(i), Pg_opt(i), Pg_opt(i)*baseMVA, Qg_opt(i), Qg_opt(i)*baseMVA);
    end
    %% Enhanced Visual Outputs for Selected Solution
    figure('Name', 'Selected Pareto Solution - IEEE 14-Bus', 'Color', [1 1 1]);
    t = tiledlayout(2, 1, 'Padding', 'compact', 'TileSpacing', 'compact');
    % Voltage Profile Plot
    nexttile;
    hBarVolt = bar(V_opt, 'FaceColor', 'flat', 'EdgeColor', 'k', 'LineWidth', 1.2);
    % Apply gradient coloring based on voltage levels
    for k = 1:length(V_opt)
        if V_opt(k) < 0.95
            hBarVolt.CData(k,:) = [0.8 0.2 0.2];  % Red tones for low voltages
        elseif V_opt(k) > 1.05
            hBarVolt.CData(k,:) = [0.2 0.2 0.8];  % Blue tones for high voltages
        else
            hBarVolt.CData(k,:) = [0.2 0.6 0.2];  % Green for acceptable voltage
        end
    end
    grid on; box on;
    xlabel('Bus Number');
    ylabel('Voltage (p.u.)');
    title('Optimal Bus Voltages Profile');
```

```
    ylim([min(V_opt)-0.05, max(V_opt)+0.05]);
    xticks(1:length(V_opt));
    % Annotate each bar with its voltage value
    for k = 1:length(V_opt)
        text(k, V_opt(k)+0.005, sprintf('%.3f', V_opt(k)), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold');
    end
    % Voltage Angle Plot
    nexttile;
    hBarAngle = bar(theta_opt*180/pi, 'FaceColor', [0.6 0.4 0.8], 'EdgeColor', 'k', 'LineWidth', 1.2);
    grid on; box on;
    xlabel('Bus Number');
    ylabel('Angle (°)');
    title('Optimal Bus Voltage Angles');
    xticks(1:length(theta_opt));
    % Annotate each angle value (in degrees)
    for k = 1:length(theta_opt)
        text(k, (theta_opt(k)*180/pi)+0.5, sprintf('%.1f°', theta_opt(k)*180/pi), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold');
    end
    % Add an overall title for the tiled layout
    title(t, 'Selected Pareto-Optimal Solution for IEEE 14-Bus System', 'FontSize', 14, 'FontWeight', 'bold');
    %% Visualize trade-offs across Pareto front
    % Compare voltage profiles and generator outputs across Pareto front
    figure('Name', 'Pareto Solutions Comparison', 'Color', [1 1 1]);
    t2 = tiledlayout(2, 1, 'Padding', 'compact', 'TileSpacing', 'compact');
    % Select solutions to compare (first, middle, last)
    compare_idx = [1, mid_idx, length(valid_cost)];
    solution_names = {'Cost-optimal', 'Mid-point', 'Voltage-optimal'};
    % Voltage profiles comparison
    nexttile;
    hold on;
    colors = [0.8 0.2 0.2; 0.2 0.6 0.2; 0.2 0.2 0.8]; % Red, Green, Blue
    markers = {'o-', 's-', 'diamond-'};
    for i = 1:length(compare_idx)
        idx = compare_idx(i);
        x_sol = valid_solutions{idx};
        V_sol = x_sol(1:nb);
        plot(1:nb, V_sol, markers{i}, 'LineWidth', 2, 'Color', colors(i,:), 'MarkerFaceColor', colors(i,:));
    end
    grid on; box on;
    xlabel('Bus Number');
    ylabel('Voltage (p.u.)');
    title('Voltage Profiles Across Pareto Front');
    xticks(1:nb);
    legend(solution_names, 'Location', 'best');
    ylim([min(Vmin)-0.02, max(Vmax)+0.02]);
    % Generator outputs comparison
    nexttile;
    gen_data = zeros(ng, length(compare_idx));
    for i = 1:length(compare_idx)
        idx = compare_idx(i);
        x_sol = valid_solutions{idx};
        Pg_sol = x_sol(2*nb+1:2*nb+ng);
        gen_data(:,i) = Pg_sol;
    end
    bar_h = bar(gen_data);
    for i = 1:length(compare_idx)
        bar_h(i).FaceColor = colors(i,:);
    end
    grid on; box on;
    xlabel('Generator Number');
    ylabel('Active Power (p.u.)');
    title('Generator Outputs Across Pareto Front');
    xticks(1:ng);
    xticklabels(arrayfun(@(x) sprintf('Gen %d', x), 1:ng, 'UniformOutput', false));
    legend(solution_names, 'Location', 'best');
    title(t2, 'Comparison of Different Solutions on the Pareto Front', 'FontSize', 14, 'FontWeight', 'bold');
end
%% Separate Objective Functions
function f = opf_cost_objective(x, nb, ng, gen_buses, a, b, c)
    % Cost-only objective
    Pg = x(2*nb+1:2*nb+ng);
    f = sum(a .* Pg.^2 + b .* Pg + c);
end
function f = opf_voltage_objective(x, nb, Vref)
    % Voltage deviation objective
    V = x(1:nb);
    f = sum((V - Vref).^2);
end
%% Nonlinear Constraints
function [c, ceq] = opf_constraints(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, Smax)
    % Extract decision variables
    V = x(1:nb);
    theta = x(nb+1:2*nb);
    Pg = x(2*nb+1:2*nb+ng);
    Qg = x(2*nb+ng+1:2*nb+2*ng);
    % Map generator outputs to their buses
    Pg_bus = zeros(nb, 1);
    Qg_bus = zeros(nb, 1);
    for i = 1:ng
        Pg_bus(gen_buses(i)) = Pg(i);
        Qg_bus(gen_buses(i)) = Qg(i);
    end
    % Power balance equations for each bus
    Pbalance = zeros(nb, 1);
    Qbalance = zeros(nb, 1);
    for i = 1:nb
        P_inj = 0;
        Q_inj = 0;
        for j = 1:nb
            angle_diff = theta(i) - theta(j);
            P_inj = P_inj + V(i) * V(j) * (G(i,j) * cos(angle_diff) + B(i,j) * sin(angle_diff));
            Q_inj = Q_inj + V(i) * V(j) * (G(i,j) * sin(angle_diff) - B(i,j) * cos(angle_diff));
        end
        Pbalance(i) = Pg_bus(i) - Pd(i) - P_inj;
        Qbalance(i) = Qg_bus(i) - Qd(i) - Q_inj;
    end
    % Line flow constraints
    line_flow = [];
    if ~isempty(Smax) && Smax(1) ~= 999
        line_flow = computeLineFlows(V, theta, nb, nl, G, B, Smax);
    end
    c = line_flow;
    ceq = [Pbalance; Qbalance];
end
%% Constraints with epsilon constraint on voltage deviation
function [c, ceq] = opf_constraints_with_epsilon(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, Smax, Vref, epsilon)
    % Regular constraints
    [c_reg, ceq] = opf_constraints(x, nb, ng, nl, gen_buses, G, B, Pd, Qd, slack_bus, Smax);
```

```matlab
    % Epsilon constraint on voltage deviation
    V = x(1:nb);
    voltage_dev = sum((V - Vref).^2);
    c_eps = voltage_dev - epsilon;
    % Combine constraints
    c = [c_reg; c_eps];
end
%% Compute Line Flows
function c = computeLineFlows(V, theta, nb, nl, G, B, Smax)
    c = [];
    line_count = 0;
    for i = 1:nb
        for j = i+1:nb
            if abs(G(i,j)) > 1e-6 || abs(B(i,j)) > 1e-6
                line_count = line_count + 1;
                if line_count <= nl
                    angle_diff = theta(i) - theta(j);
                    Pij = V(i)^2 * G(i,j) - V(i)*V(j) * (G(i,j)*cos(angle_diff) + B(i,j)*sin(angle_diff));
                    Qij = -V(i)^2 * B(i,j) - V(i)*V(j) * (G(i,j)*sin(angle_diff) - B(i,j)*cos(angle_diff));
                    Sij = sqrt(Pij^2 + Qij^2);
                    c = [c; Sij - Smax(line_count)];
                end
            end
        end
    end
end
%% Extract Weights from Pareto Front
function extractWeightsFromPareto(valid_cost, valid_volt, valid_solutions)
    fprintf('\n----- Weights Analysis from Pareto Front -----\n');
    fprintf('Point | Cost Value | Voltage Dev | w1 (Cost) | w2 (Voltage) | w1/w2 Ratio\n');
    fprintf('-------------------------------------------------------------------------\n');
    % Normalize the objectives to similar scales for fair comparison
    max_cost = max(valid_cost);
    min_cost = min(valid_cost);
    max_volt = max(valid_volt);
    min_volt = min(valid_volt);
    norm_cost = (valid_cost - min_cost) / (max_cost - min_cost);
    norm_volt = (valid_volt - min_volt) / (max_volt - min_volt);
    % Calculate weights for each point based on the local gradient
    for i = 1:length(valid_cost)
        if i == 1
            % First point - use forward difference
            dc = norm_cost(i+1) - norm_cost(i);
            dv = norm_volt(i+1) - norm_volt(i);
        elseif i == length(valid_cost)
            % Last point - use backward difference
            dc = norm_cost(i) - norm_cost(i-1);
            dv = norm_volt(i) - norm_volt(i-1);
        else
            % Interior points - use central difference
            dc = norm_cost(i+1) - norm_cost(i-1);
            dv = norm_volt(i+1) - norm_volt(i-1);
        end
        % Calculate gradient (slope of tangent line)
        if abs(dc) < 1e-10
            w1 = 0;
            w2 = 1;
        elseif abs(dv) < 1e-10
            w1 = 1;
            w2 = 0;
        else
            % Gradient of the Pareto front at this point
            gradient = -dv/dc;
            % Convert gradient to weights (w1 for cost, w2 for voltage)
            w1 = 1 / (1 + gradient);
            w2 = gradient / (1 + gradient);
        end
        % Normalize weights to sum to 1
        sum_w = w1 + w2;
        w1 = w1 / sum_w;
        w2 = w2 / sum_w;
        if w2 < 1e-10
            ratio = "inf";
        else
            ratio = w1/w2;
        end
        % Print results
        fprintf('%4d | %10.4f | %10.6f | %9.4f | %12.4f | %10s\n', ...
            i, valid_cost(i), valid_volt(i), w1, w2, num2str(ratio));
    end
    [~, balanced_idx] = min(abs(arrayfun(@(i) (norm_cost(i) - norm_volt(i)), 1:length(valid_cost))));
    normalized_sum = norm_cost + norm_volt;
    [~, min_sum_idx] = min(normalized_sum);
    fprintf('\nRecommended Weight Sets:\n');
    fprintf('1. Most balanced weights: w1 = %.4f, w2 = %.4f (point %d)\n', ...
        1/(1+norm_volt(balanced_idx)/norm_cost(balanced_idx)), ...
        1/(1+norm_cost(balanced_idx)/norm_volt(balanced_idx)), balanced_idx);
     mid_idx = ceil(length(valid_cost) / 2);
    if mid_idx == balanced_idx
        fprintf('2. Middle Pareto point: w1 = %.4f, w2 = %.4f (point %d) (same as balanced)\n', ...
            1/(1+norm_volt(mid_idx)/norm_cost(mid_idx)), ...
            1/(1+norm_cost(mid_idx)/norm_volt(mid_idx)), mid_idx);
    else
        fprintf('2. Middle Pareto point: w1 = %.4f, w2 = %.4f (point %d)\n', ...
            1/(1+norm_volt(mid_idx)/norm_cost(mid_idx)), ...
            1/(1+norm_cost(mid_idx)/norm_volt(mid_idx)), mid_idx);
    end
    if min_sum_idx == balanced_idx || min_sum_idx == mid_idx
       else
        fprintf('3. Minimum normalized objective sum: w1 = %.4f, w2 = %.4f (point %d)\n', ...
            1/(1+norm_volt(min_sum_idx)/norm_cost(min_sum_idx)), ...
            1/(1+norm_cost(min_sum_idx)/norm_volt(min_sum_idx)), min_sum_idx);
    end
    figure('Name', 'Weight Distribution Along Pareto Front', 'Color', [1 1 1]);
    w1_values = zeros(length(valid_cost), 1);
    w2_values = zeros(length(valid_cost), 1);
    for i = 1:length(valid_cost)
        if i == 1
            dc = norm_cost(i+1) - norm_cost(i);
            dv = norm_volt(i+1) - norm_volt(i);
        elseif i == length(valid_cost)
            dc = norm_cost(i) - norm_cost(i-1);
            dv = norm_volt(i) - norm_volt(i-1);
        else
            dc = norm_cost(i+1) - norm_cost(i-1);
            dv = norm_volt(i+1) - norm_volt(i-1);
        end
       % Calculate weights
        if abs(dc) < 1e-10
            w1_values(i) = 0;
            w2_values(i) = 1;
        elseif abs(dv) < 1e-10
            w1_values(i) = 1;
            w2_values(i) = 0;
        else
            gradient = -dv/dc;
            w1_values(i) = 1 / (1 + gradient);
            w2_values(i) = gradient / (1 + gradient);
```

```
            % Normalize
            sum_w = w1_values(i) + w2_values(i);
            w1_values(i) = w1_values(i) / sum_w;
            w2_values(i) = w2_values(i) / sum_w;
        end
    end

    % Plotting weights
    plot(1:length(valid_cost), w1_values, 'r-o', 'LineWidth', 2, 'MarkerFaceColor', 'r');
    hold on;
    plot(1:length(valid_cost), w2_values, 'b-s', 'LineWidth', 2, 'MarkerFaceColor', 'b');
    plot([balanced_idx balanced_idx], [0 1], 'k--', 'LineWidth', 1.5);
    plot([mid_idx mid_idx], [0 1], 'g--', 'LineWidth', 1.5);
    grid on; box on;
    xlabel('Point Index on Pareto Front');
    ylabel('Weight Value');
    title('Weight Distribution Along Pareto Front');
    legend('w1 (Cost)', 'w2 (Voltage)', 'Balanced Point', 'Middle Point', 'Location', 'best');
    ylim([0 1]);
end
```

branch_data.xlsx

1	From_Bus	To_Bus	G_pu	B_pu	S_max_MVA
2	1	2	4.999131601	-15.26308652	0
3	1	5	1.025897455	-4.234983682	0
4	2	3	1.135019192	-4.781863152	0
5	2	4	1.686033151	-5.115838326	0
6	2	5	1.701139667	-5.193927398	0
7	3	4	1.98597571	-5.068816978	0
8	4	5	6.840980661	-21.57855398	0
9	4	7	0	-4.781943382	0
10	4	9	0	-1.797979072	0
11	5	6	0	-3.967939052	0
12	6	11	1.955028563	-4.094074344	0
13	6	12	1.52596744	-3.175963965	0
14	6	13	3.098927404	-6.102755448	0
15	7	8	0	-5.676979847	0
16	7	9	0	-9.09008272	0
17	9	10	3.902049552	-10.36539413	0
18	9	14	1.424005487	-3.029050457	0
19	10	11	1.880884754	-4.402943749	0
20	12	13	2.489024587	-2.251974626	0
21	13	14	1.136994158	-2.314963475	0

bus_data.xlsx

1	Bus	P_Load_MW	Q_Load_MVAr	V_ref_pu
2	1	0	0	1.06
3	2	21.7	12.7	1.045
4	3	94.2	19	1.01
5	4	47.8	-3.9	1.019
6	5	7.6	1.6	1.02
7	6	11.2	7.5	1.07
8	7	0	0	1.062
9	8	0	0	1.09
10	9	29.5	16.6	1.056
11	10	9	5.8	1.051
12	11	3.5	1.8	1.057
13	12	6.1	1.6	1.055
14	13	13.5	5.8	1.05
15	14	14.9	5	1.036

Plagiarism Test on QueText: (since Turnitin was not available)

Due to limit,we divided the document into two parts and conducted the plagiarism test on them and these are the reports of the two parts: