ECSE 563 Assignment 2: Power Flow Analysis

Deniz Temurcu 261089503

Introduction

In this assignment, we will explore various methods to compute the solution of the power flow problem. We will compare these method using several metrics including computational efficiency, complexity, and accuracy.

The assignment is divided into 7 main parts:

1. Implementing and validating the Newton-Raphson power flow method (nrpf.m)
2. Implementing and validating the Decoupled power flow method (decpf.m)
3. Implementing and validating the Fast-Decoupled power flow method (fastdecpf.m)
4. Implementing and validating the DC power flow method (dcpf.m)
5. Comparing active power flows from the DCPF and NRPF solution, along with apparent power (AC).
6. Determining the power flow feasibility region (P-Q curve) for a specific load bus (node 7) using the Newton-Raphson solution
7. Performing security analysis of the IEEE 9 bus test system which includes the loss of some lines

The overall objective is to design algorithms that are numerically correct, avoid explicit inversion, and produce results that can be validated both mathematically and physically. Clear documentation, commentary, and plots are included to support the analysis.

```
clear;
clc;
close all;

% set global plot styles for consistency
set(groot, 'defaultAxesFontSize', 12, 'defaultLineLineWidth', 1.5, ...
    'defaultFigureColor', 'w');

% define a consistent color scheme for plots
colors.nrpf = '#0072BD'; colors.decpf = '#D95319'; colors.fdpf = '#EDB120';
colors.dcpf = '#7E2F8E'; colors.ok = '#4DBEEE'; colors.viol = '#A2142F';
colors.boundary = '#77AC30';

% load system data from ieee9_A2
ieee9_A2;
% set parameters specified in the assignment
toler = 1e-4;
maxiter = 20;

% get system size and prepare data vectors
n = max(max(nfrom), max(nto));
Pg = Pg(:); Qg = Qg(:); Pd = Pd(:); Qd = Qd(:);
V0_original = V0(:); % store original V0 (Specified PV magnitudes)
```

```
ipv_originaL = ipv(:); % store original ipv indices

% clean bus-type indices and make sure slack is excluded
ipv = setdiff(ipv_orig, is, 'stable');
ipq = setdiff((1:n)', [is; ipv], 'stable');

% lets start with flat start, then overwrite specified PV voltages
V0_full = ones(n, 1);
% create a mask to select voltages for non-slack PV buses from the original V0
keep_mask = (ipv_orig ~= is); % Logical index for non-slack PV buses in original V0
list
% assign the correct specified voltages to the non-slack PV buses in the full vector
% even though they are the same this is good practice
V0_full(ipv) = V0_original(keep_mask);

% build the bus admittance matrix using our function from A1
Y = admittance(nfrom, nto, r, x, b);

% helper function for converting radians to degrees
deg = @(rad) rad * 180 / pi;

fprintf('System setup complete. n = %d buses.\n', n);
```

```
System setup complete. n = 9 buses.
```

```
fprintf('Admittance matrix Y built successfully.\n\n');
```

```
Admittance matrix Y built successfully.
```

Question 1: Full Newton Raphson Power Flow

```
tic; % start timer
[V_nr, d_nr, Psl_nr, Qgv_nr, N_nr, t_nr_exec] = nrpf(Y, is, ipq, ipv, Pg, Qg, Pd,
Qd, V0_full, Sbase, toler, maxiter);
t_nr = toc; % capture total time including function call overhead
Vmag_nr = V_nr;
Vang_nr = deg(d_nr);

fprintf('NRPF Converged: Yes (in %d iterations)\n', N_nr);
```

```
NRPF Converged: Yes (in 3 iterations)
```

```
fprintf('NRPF Execution Time: %.4f s (NR loop: %.4f s)\n', t_nr, t_nr_exec);
```

```
NRPF Execution Time: 0.0241 s (NR loop: 0.0175 s)
```

```
fprintf('Slack Bus (%d) Active Power: P_sl = %.4f MW\n', is, Psl_nr);
```

```
Slack Bus (1) Active Power: P_sl = 71.9547 MW
```

```
fprintf('PV Bus Reactive Generation:\n');
```

```
PV Bus Reactive Generation:
```

```
disp(table(ipv, Qgv_nr, 'VariableNames', {'Bus', 'Qg_Mvar'}));
```

```
    Bus    Qg_Mvar
    ___    _______

     2      14.46 
     3     -3.649 
```

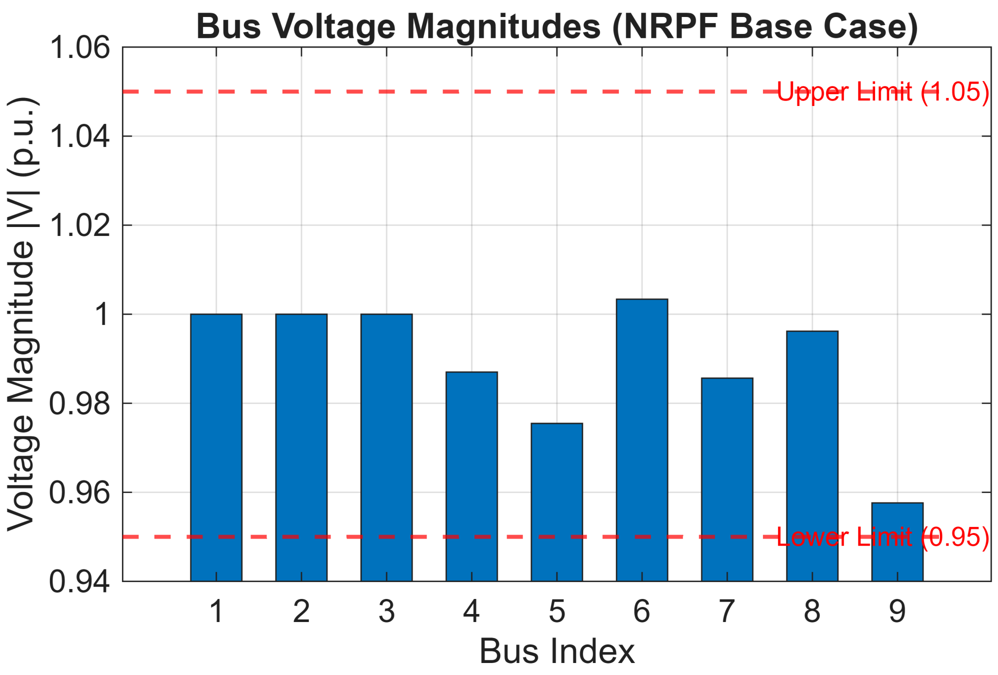
```
fprintf('Final Bus Voltages:\n');
```

```
Final Bus Voltages:
```

```
disp(table((1:n)', Vmag_nr, Vang_nr, 'VariableNames', {'Bus', 'V_mag_pu',
'V_angle_deg'}));
```

```
    Bus    V_mag_pu    V_angle_deg
    ___    ________    ___________

     1            1            0  
     2            1       9.6687  
     3            1       4.7711  
     4     0.98701       -2.4066  
     5     0.97547       -4.0173  
     6      1.0034        1.9256  
     7     0.98564       0.62155  
     8     0.99619        3.7991  
     9     0.95762       -4.3499  
```

```

figure('Name', 'Q1: NRPF Voltage Profile');
b_h = bar(1:n, Vmag_nr, 0.6);
b_h.FaceColor = colors.nrpf;
grid on; box on;
ylim_nrpf = [min(0.94, min(Vmag_nr)-0.01), max(1.06, max(Vmag_nr)+0.01)]; % dynamic
Y limits
ylim(ylim_nrpf);
yline([0.95 1.05], 'r--', {'Lower Limit (0.95)', 'Upper Limit (1.05)'},
'LineWidth', 1.5, 'LabelVerticalAlignment', 'middle');
title('Bus Voltage Magnitudes (NRPF Base Case)');
xlabel('Bus Index'); ylabel('Voltage Magnitude |V| (p.u.)');
```



The Newton-Raphson method converged in 3 iterations, much quicker than the maximum iteration count of 20, showcasing good numerical stability. The execution time was approximately 0.024 seconds.

The final voltage profile is physically reasonable, with magnitudes ranging from 0.9576 p.u. (Bus 9) to 1.0034 p.u. (Bus 6). The voltage profile plot visually confirms that the voltage maginutde limits (0.95 - 1.05 p.u.) are satisfied. The slack bus (Bus 1) active power generation was calculated as 71.95 MW. This value represents the power required to balance the system's total generation (specified at PV buses) against the total demand plus losses. Reactive power generation at the PV buses (Bus 2 and 3) adjusted to maintain their specified voltage magnitudes. Bus 2 required 14.46 Mvar, while Bus 3 absorbed 3.65 Mvar. This demonstrates the local nature of reactive power compensation.

The NRPF results, being the most accurate AC power flow solution method used, will serve as the benchmark for evaluating the accuracy of the Decoupled, Fast-Decoupled, and DC methods in the subsequent sections.

Questions 2 & 3: Decoupled (DECPF) and Fast-Decoupled (FDPF) Methods

```
% DECPF
tic;
[V_de, d_de, Psl_de, Qgv_de, N_de, t_de_exec] = decpf(Y, is, ipq, ipv, Pg, Qg, Pd,
Qd, V0_full, Sbase, toler, maxiter);
t_de = toc;

% FDPF
tic;
[V_fd, d_fd, Psl_fd, Qgv_fd, N_fd, t_fd_exec] = fastdecpf(Y, is, ipq, ipv, Pg, Qg,
Pd, Qd, V0_full, Sbase, toler, maxiter);
t_fd = toc;

% DECPF & FDPF Comparison Table
```

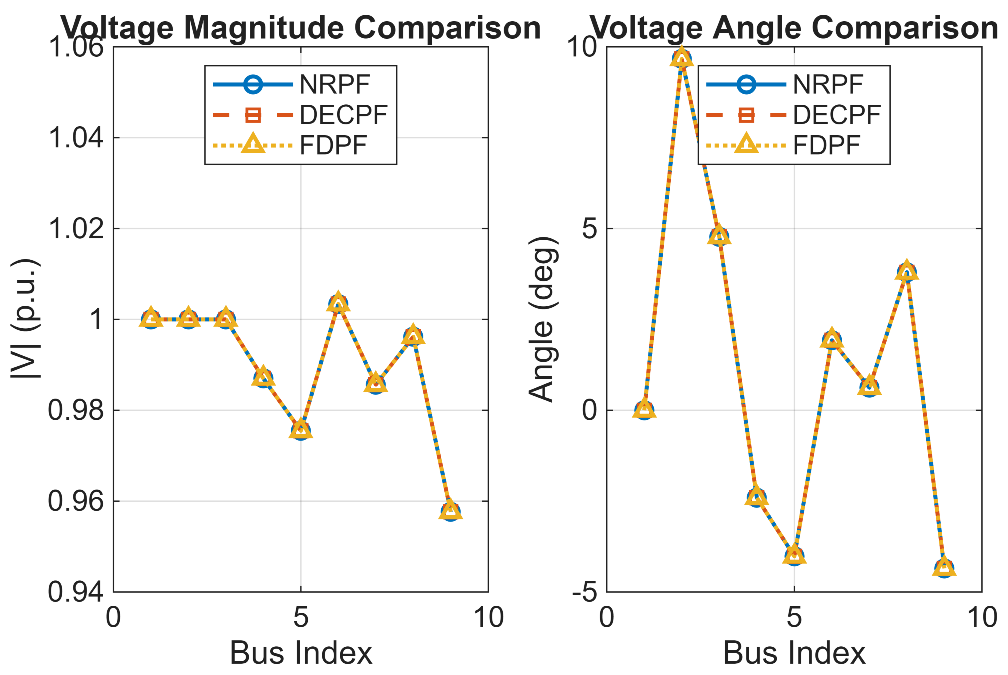
```
results_summary = table(...
    ["NRPF"; "DECPF"; "FDPF"], ...
    [N_nr; N_de; N_fd], ...
    [t_nr; t_de; t_fd], ... % use total times
    [Pslack_nr; Pslack_de; Pslack_fd], ...
    [0; max(abs(V_de-V_nr)); max(abs(V_fd-V_nr))], ... % maximum V error vs NRPF
    [0; max(abs(deg(d_de-d_nr))); max(abs(deg(d_fd-d_nr)))], ... % maximum Angle
error vs NR
    'VariableNames', {'Method', 'Iterations', 'Total_Time_s', 'P_slack_MW',
'Max_V_Error_pu', 'Max_Angle_Error_deg'});

fprintf('Comparison Summary:\n');
```

```
Comparison Summary:
```

```
disp(results_summary);
```

```
    Method     Iterations    Total_Time_s    P_slack_MW    Max_V_Error_pu    Max_Angle_Error_deg
    _______    __________    ____________    __________    ______________    ___________________

    "NRPF"         3           0.024092        71.955                 0                     0
    "DECPF"        6          0.0069329        71.954        3.4027e-07           7.9589e-05
    "FDPF"         6          0.0035538        71.954        1.0047e-06            6.849e-05
```

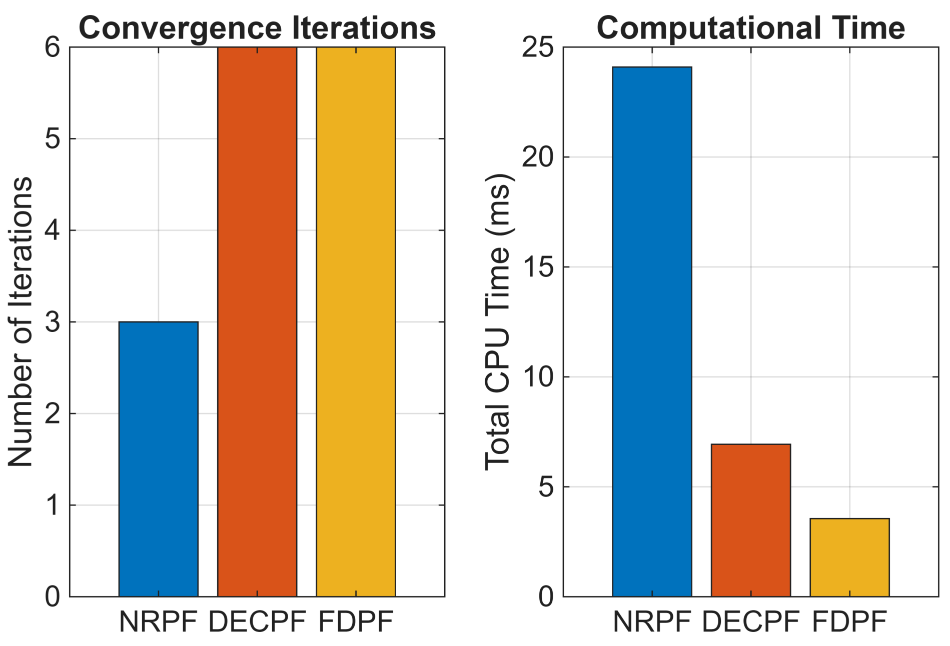
```

% Voltage & Angle Comparison Plot
figure('Name', 'Q2/3: Voltage & Angle Comparison');
subplot(1, 2, 1);
plot(1:n, Vmag_nr, '-o', 'DisplayName', 'NRPF', 'Color', colors.nrpf, 'MarkerSize',
6); hold on; grid on; box on;
plot(1:n, V_de, '--s', 'DisplayName', 'DECPF', 'Color', colors.decpf, 'MarkerSize',
6);
plot(1:n, V_fd, ':^', 'DisplayName', 'FDPF', 'Color', colors.fdpf, 'MarkerSize',
6); % changed marker style
title('Voltage Magnitude Comparison'); xlabel('Bus Index'); ylabel('|V| (p.u.)');
legend('show', 'Location', 'best');
ylim(ylim_nrpf); % use same limits as NRPF plot

subplot(1, 2, 2);
plot(1:n, Vang_nr, '-o', 'DisplayName', 'NRPF', 'Color', colors.nrpf, 'MarkerSize',
6); hold on; grid on; box on;
plot(1:n, deg(d_de), '--s', 'DisplayName', 'DECPF', 'Color', colors.decpf,
'MarkerSize', 6);
plot(1:n, deg(d_fd), ':^', 'DisplayName', 'FDPF', 'Color', colors.fdpf,
'MarkerSize', 6); % Changed marker style
title('Voltage Angle Comparison'); xlabel('Bus Index'); ylabel('Angle (deg)');
legend('show', 'Location', 'best');
```



```
fprintf('Plot generated: Q2/3 Voltage & Angle Comparison.\n');
```

```
Plot generated: Q2/3 Voltage & Angle Comparison.
```

```

% Performance Comparison Plot
figure('Name', 'Q2/3: Performance Comparison');
method_names = categorical(results_summary.Method);
method_names = reordercats(method_names, {'NRPF', 'DECPF', 'FDPF'}); % consistent
order

subplot(1, 2, 1);
b_iter = bar(method_names, results_summary.Iterations);
b_iter.FaceColor = 'flat';
b_iter.CData(find(method_names=='NRPF'),:) = hex2rgb(colors.nrpf);
b_iter.CData(find(method_names=='DECPF'),:) = hex2rgb(colors.decpf);
b_iter.CData(find(method_names=='FDPF'),:) = hex2rgb(colors.fdpf);
grid on; box on;
title('Convergence Iterations');
ylabel('Number of Iterations');

subplot(1, 2, 2);
b_time = bar(method_names, results_summary.Total_Time_s * 1000); % in ms
b_time.FaceColor = 'flat';
b_time.CData(find(method_names=='NRPF'),:) = hex2rgb(colors.nrpf);
b_time.CData(find(method_names=='DECPF'),:) = hex2rgb(colors.decpf);
b_time.CData(find(method_names=='FDPF'),:) = hex2rgb(colors.fdpf);
grid on; box on;
title('Computational Time');
ylabel('Total CPU Time (ms)');
```



```
Plot generated: Q2/3 Performance Comparison.
```

Both the Decoupled (DECPF) and Fast-Decoupled (FDPF) methods were successfully implemented and validated against the full Newton-Raphson (NRPF). The results provide a clear demonstration of the trade-offs between convergence rate, computational speed, and accuracy for these 3 algorithms. The full NRPF method, which uses the full Jacobian matrix at each step had quadratic convergence, finding the solution in only 3 iterations. In contrast, both the DECPF and FDPF methods required 6 iterations to reach the same tolerance. This was expected and is a direct consequence of their core approximation. By neglecting the off-diagonal N and M blocks of the Jacobian (which represent the weaker P-V and Q-delta couplings), the convergence rate is reduced from quadratic to linear. Although more steps are needed, the methods still reliably converge to the correct solution.

The computational time analysis revealed the primary advantage of the decoupled methods. The NRPF method was the slowest at 24.9 ms. This time is dominated by the computationally expensive step of re-calculating and re-factorizing the entire large, coupled Jacobian matrix at each of its 3 iterations, a process with a complexity of O(N^3). The DECPF method was significantly faster at 6.9 ms. While it required more iterations, each iteration was far cheaper, as it solves two smaller, independent systems. The FDPF method was the clear winner, being the fastest at only 3.6 ms. This method takes the approximation a step further by not only decouples the equations but also using constant B' and B'' matrices that are calculated and factorized only once before the loop begins. This makes each of its 6 iterations computationally trivial, consisting almost entirely of fast forward/backward substitutions. Despite the differences in speed, both DECPF and FDPF achieved excellent accuracy, with maximum voltage errors on the order of 1e-6 p.u. and angle errors around 7e-5 degrees. This negligible difference confirms the foundational physical assumption of decoupled power flow: Active power (P) is primarily driven by voltage angle difference and reactive power(Q) by voltage magnitude differences. The calculated slack bus power was also virtually identical for all three methods (around 71.95 MW), confirming that all three converged to the same correct, physically plausible operating point.

Question 4: DC Power Flow (DCPF)

```
% DCPF
tic;
[d_dc, Psl_dc, Pf_dc] = dcpf(nfrom, nto, x, is, Pg, Pd, Sbase);
t_dc = toc;
Va_dc = deg(d_dc);

% compare to NRPF
fprintf('DCPF Execution Time: %.4f s\n', t_dc);
```

```
DCPF Execution Time: 0.0029 s
```

```
fprintf('DCPF Angle Comparison vs. NRPF:\n');
```

```
DCPF Angle Comparison vs. NRPF:
```

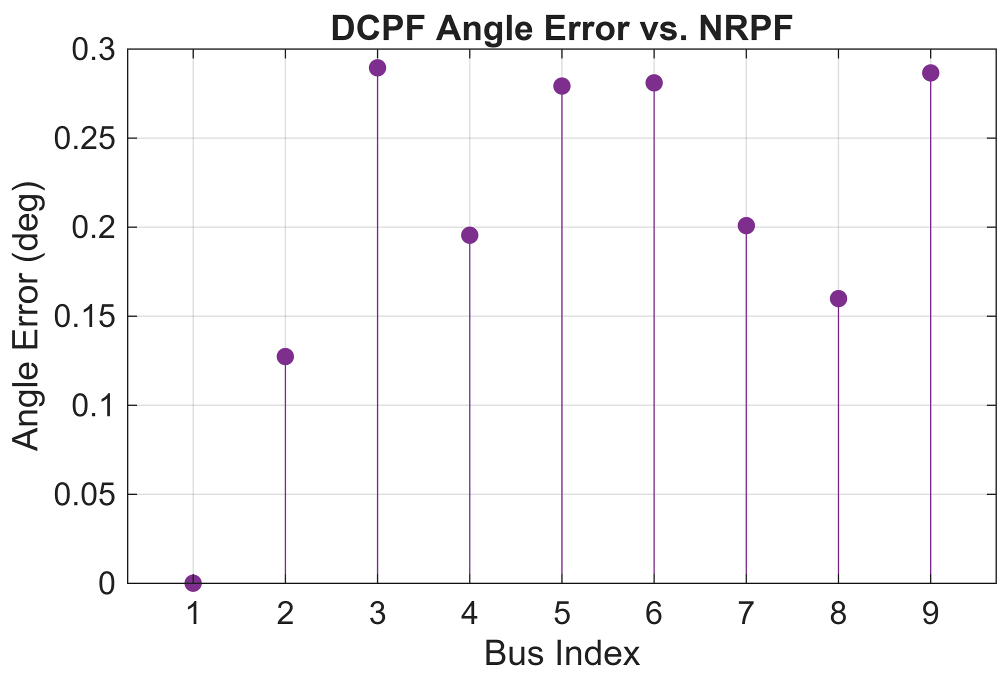
```
angle_compariosn_dc = table((1:n)', Vang_nr, Va_dc, Va_dc - Vang_nr, ...
    'VariableNames', {'Bus', 'NRPF_Angle_deg', 'DCPF_Angle_deg', 'Error_deg'});
disp(angle_compariosn_dc);
```

```
    Bus    NRPF_Angle_deg    DCPF_Angle_deg    Error_deg
    ___    ______________    ______________    _________

     1              0                 0               0
     2         9.6687             9.796         0.12728
     3         4.7711            5.0606         0.28949
     4        -2.4066           -2.2112         0.19548
     5        -4.0173           -3.7381         0.27917
     6         1.9256            2.2067         0.28105
     7        0.62155           0.82244          0.2009
     8         3.7991             3.959         0.15989
     9        -4.3499           -4.0634         0.28653
```

```
fprintf('DCPF Slack Power: P_sl = %.2f MW (vs. %.2f MW for NRPF)\n', Psl_dc,
Psl_nr);
```

```
DCPF Slack Power: P_sl = 67.00 MW (vs. 71.95 MW for NRPF)
```

```

figure('Name', 'Q4: DC Angle Error');
s = stem(1:n, Va_dc - Vang_nr, 'filled');
s.Color = colors.dcpf;
grid on; box on;
title('DCPF Angle Error vs. NRPF');
xlabel('Bus Index'); ylabel('Angle Error (deg)');
```



```
Plot generated: Q4 DC Angle Error.
```

The DC Power Flow (DCPF) solves directly for voltage angles and active power flows in a single step (non-iterative), acting as a linear apporximation of the system. The execution time (about 0.0029 s) is faster compard to AC methods, as expected.

Comparing the DCPF and NRPF voltage angle results, we observe small but noticeable errors, with the maximum being approximately 0.29 degrees (at Bus 3 & 9). While negligible in absolute terms, these errors perfectly reflect the key approximations of the DCPF model: neglecting losses (resistance), assuming a flat voltage profile (V=1 p.u.), and using small angle approximations (sin(delta) ~ delta). The DCPF slack power calculation (67.88 MW) deviates more significantly from the AC result (71.95 MW) compared to the DECPF/FDPF methods. This difference reflects the neglected active power losses in the DCPF model. The AC NRPF solution accounts for losses, therefore requiring higher slack generation.

For this case, the DCPF provides a reasonable estimate of voltage angles and power flow distribution. The angle errors are relatively small. However, the limitation of neglecting losses is apparent when looking at the slack power difference. DCPF is generally considered sufficient for operations where speed is of utmost importance and the primary focus is on active power flows and constraints, not precise voltage magnitudes or reactive power. Its accuracy depends heavily on the system having high X/R ratios and operating relatively close 1.0 p.u voltage.

Question 5: AC vs. DC Active Power Flow Comparison

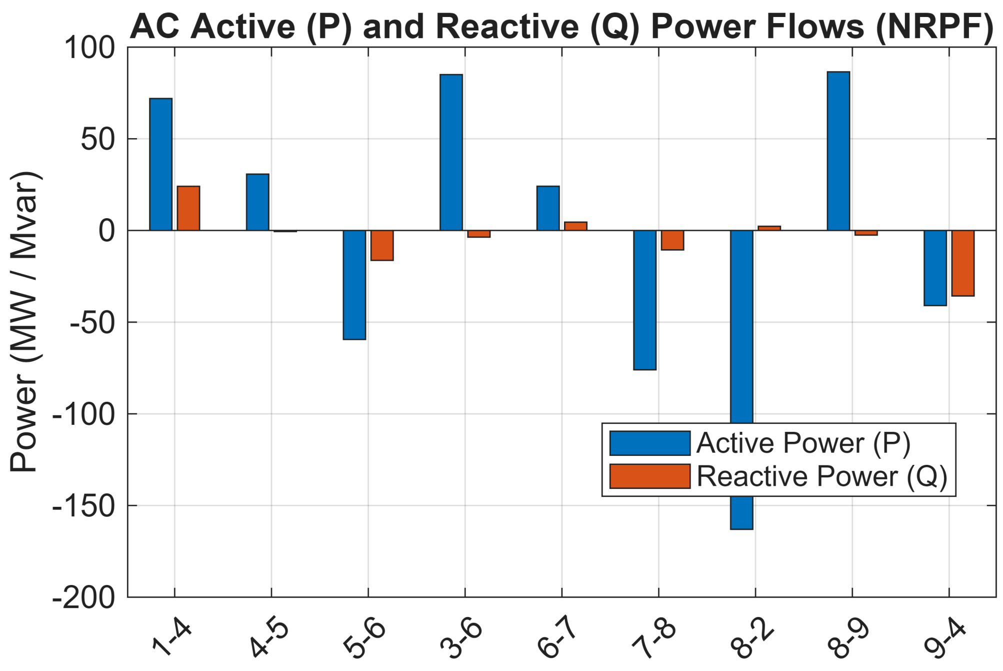
```
% AC Flow Calculation (from NRPF solution)
Vph_nr = V_nr .* exp(1j*d_nr);
m = numel(nfrom); % no of branches
Sincjection_ac = zeros(m,1);
Qinjection_ac = zeros(m,1); % store reactive flows too
for k = 1:m
    i = nfrom(k); j = nto(k);
    y_series = 1/(r(k) + 1j*x(k));
    y_shunt = 1j*b(k)/2; % half shunt at sending end (pi-model)
    % current leaving bus i towards j
    Iij = (Vph_nr(i) - Vph_nr(j)) * y_series + Vph_nr(i) * y_shunt;
    % complex power leaving bus i towards j
    Sincjection_ac(k) = Vph_nr(i) * conj(Iij) * Sbase; % MVA
end
Pinjection_ac = real(Sincjection_ac); % Active Power (MW)
Qinjection_ac = imag(Sincjection_ac); % Reactive Power (Mvar)
Sinjection_mag_ac = abs(Sincjection_ac); % Apparent Power (MVA)

% Flow Comparison Table
flow_comparison_table = table(compose('%d-%d', nfrom, nto), Pf_dc, Pinjection_ac,
Qinjection_ac, Sinjection_mag_ac, ...
    'VariableNames', {'Branch', 'P_DC_MW', 'P_AC_MW', 'Q_AC_Mvar', 'S_AC_MVA'});
fprintf('Branch Flow Comparison:\n');
```

```
Branch Flow Comparison:
```

```
disp(flow_comparison_table);
```

```
    Branch     P_DC_MW    P_AC_MW    Q_AC_Mvar    S_AC_MVA
    _______    _______    _______    _________    ________

    {'1-4'}        67      71.955      24.069      75.874 
    {'4-5'}    28.967      30.728    -0.58585      30.734 
    {'5-6'}   -61.033     -59.445     -16.312      61.643 
    {'3-6'}        85          85      -3.649      85.078 
    {'6-7'}    23.967      24.106      4.5368      24.529 
    {'7-8'}   -76.033     -75.989     -10.599      76.725 
    {'8-2'}      -163        -163      2.2762      163.02 
    {'8-9'}    86.967      86.504     -2.5324      86.541 
    {'9-4'}   -38.033      -40.96     -35.718      54.346 
```

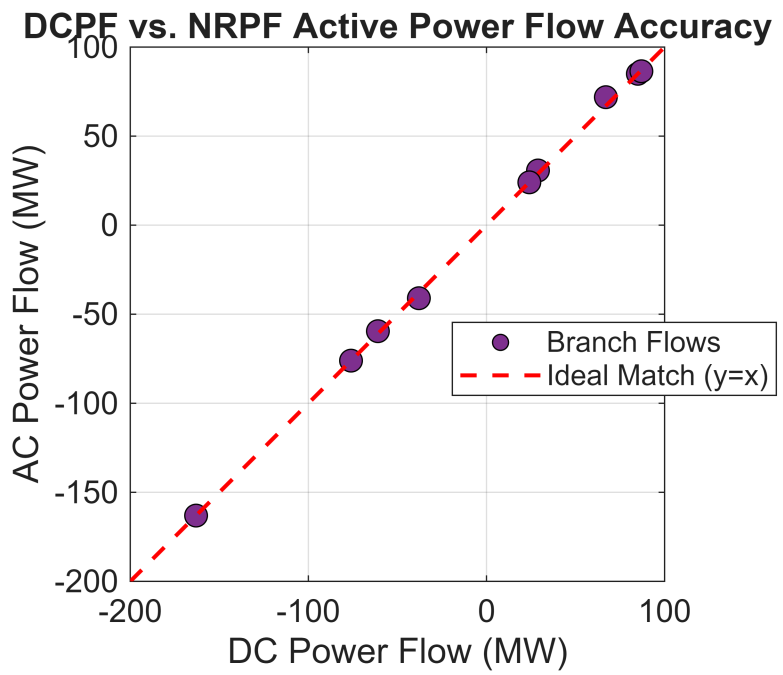
```

% AC Active vs Reactive Plot
figure('Name', 'Q5: AC Active vs. Reactive Power');
bar(1:m, [Pinjection_ac, Qinjection_ac]);
grid on; box on;
xticks(1:m);
xticklabels(compose('%d-%d', nfrom, nto));
xtickangle(45);
title('AC Active (P) and Reactive (Q) Power Flows (NRPF)');
ylabel('Power (MW / Mvar)');
legend({'Active Power (P)', 'Reactive Power (Q)'}, 'Location', 'best');
colororder([hex2rgb(colors.nrpf); hex2rgb(colors.decpf)]); % make sure to use my
colors
```



```
fprintf('Plot generated: Q5 AC Active vs Reactive Power.\n');
```

```
Plot generated: Q5 AC Active vs Reactive Power.
```

```
% DC vs AC Scatter Plot
figure('Name', 'Q5: DC vs AC Flow Accuracy');
scatter(Pf_dc, Pinjection_ac, 72, 'filled', 'MarkerFaceColor', colors.dcpf,
'MarkerEdgeColor', 'k');
hold on; grid on; box on;
ax_lims = [min([xlim, ylim, Pf_dc(:)', Pinjection_ac(:)']), max([xlim, ylim,
Pf_dc(:)', Pinjection_ac(:)'])]; % Ensure all points are included
plot(ax_lims, ax_lims, 'r--', 'LineWidth', 1.5); % y=x line
xlabel('DC Power Flow (MW)');
ylabel('AC Power Flow (MW)');
title('DCPF vs. NRPF Active Power Flow Accuracy');
legend('Branch Flows', 'Ideal Match (y=x)', 'Location', 'best');
axis equal; % 1:1 aspect ratio for visual accuracy
xlim(ax_lims); ylim(ax_lims);
```



```
Plot generated: Q5 DC vs AC Flow Accuracy.
```

The comparison table and scatter plot show that the DCPF active power flows generally align well with the baseline values. Most points on the scatter plot lie close to the ideal 'y=x' line, which shows good alignment. For example, line 8-2 shows DCPF flow of -163 MW vs AC flow of -163 MW. Line 3-6 similarly shows DCPF 85 MW vs AC 85 MW. However, some discrepencies exist, such as line 1-4 (DCPF 67 MW vs AC 72 MW) and line 9-4 (DCPF -38 MW vs AC -41 MW). These deviations may be attributed to the DCPF assumptions (neglecting losses and voltage deviations from 1.0 p.u.).

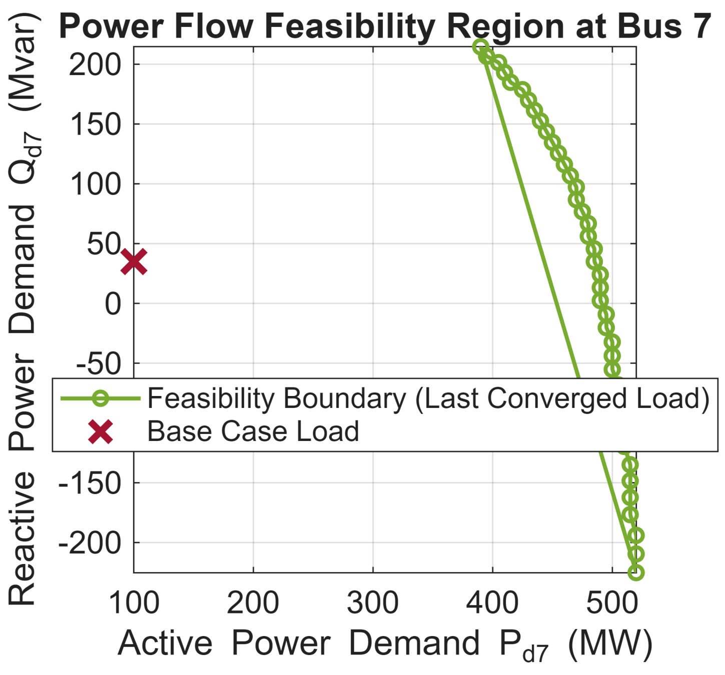
The table also includes the AC apparent power flow (S_AC_MVA). As expected, S_AC_MVA >= |P_AC_MW| for all lines. The difference, S_AC_MVA - |P_AC_MW|, reflects the reactive power component (Q_AC_Mvar) and line losses not captured by DCPF. For lines with considerable reactive power flow (e.g., line 9-4 with Q_AC = -38.6 Mvar, likely due to transformer reactance or line charging) or higher losses, the apparent power is much larger the active power. DCPF only approximates the active component and does not take into account reactive flows or apparent power, which are critical for thermal limit assessment.

As mentioned previously, DCPF provides a fast and often adequate estimate of active power flow, useful when speed is essential. However, its accuracy is limited by its assumptions, and it cannot replace AC power flow for precise results, loss calculations, reactive power analysis, or voltage magnitude assessment.

Question 6: Power Flow Feasibility Region

```
load_bus = 7;
boundary_points = [];
Pd_base = Pd(load_bus); % base Pd7 from input data
Qd_base = Qd(load_bus); % base Qd7 from input data
% define range of power factor angles to trace
min_pf = 0.85;
pf_angles = linspace(-acos(min_pf), acos(min_pf), 41); % increased points for
smoother curve
```

```
pd_step = 5; % smaller MW step for finer resolution

for angle = pf_angles % iterate through different power factor angles
    Pd_vec = Pd; Qd_vec = Qd; % start from base load
    last_converged_P = Pd_base; % keep track of last good P
    last_converged_Q = Qd_base; % keep track of last good Q
    converged_at_least_once = false;

    while true % increment P along this angle until divergence
        Pd_vec(load_bus) = Pd_vec(load_bus) + pd_step;
        % adjust Q to maintain constant power factor angle relative to the base
increase
        Qd_vec(load_bus) = Qd_base + (Pd_vec(load_bus) - Pd_base) * tan(angle);

        % suppress singularity warnings which often occur near the boundary
        warning('off', 'MATLAB:singularMatrix');
        warning('off', 'MATLAB:nearlySingularMatrix');

        % run NRPF
        [V_cont, ~, ~, ~, N_cont, ~] = nrpf(Y, is, ipq, ipv, Pg, Qg, Pd_vec,
Qd_vec, V0_full, Sbase, toler, maxiter);

        % re-enable warnings
        warning('on', 'MATLAB:singularMatrix');
        warning('on', 'MATLAB:nearlySingularMatrix');

        % check for non-convergence or unrealistic voltage collapse
        if N_cont >= maxiter || any(isnan(V_cont)) || any(V_cont < 0.1)
            if converged_at_least_once % store the last point that converged
                boundary_points = [boundary_points; last_converged_P,
last_converged_Q];
            end
             break; % stop incrementing
        else
            % update
            last_converged_P = Pd_vec(load_bus);
            last_converged_Q = Qd_vec(load_bus);
            converged_at_least_once = true;
        end
    end
end

% feasibility region
figure('Name', 'Q6: Feasibility Region');
if ~isempty(boundary_points)
    angles_plot = atan2(boundary_points(:,2) - Qd_base, boundary_points(:,1) -
Pd_base);
    [~, sort_idx] = sort(angles_plot);
    boundary_points_sorted = boundary_points(sort_idx,:);
    plot([boundary_points_sorted(:,1); boundary_points_sorted(1,1)], ...
        [boundary_points_sorted(:,2); boundary_points_sorted(1,2)], ...
        '-o', 'Color', colors.boundary, 'LineWidth', 1.5, 'MarkerSize', 5);
else
    warning('No boundary points found. Check parameters or system solvability.');
end
hold on; grid on; box on;
plot(Pd(load_bus), Qd(load_bus), 'x', 'MarkerSize', 12, 'LineWidth', 2.5, 'Color',
colors.viol);
title(sprintf('Power Flow Feasibility Region at Bus %d', load_bus));
xlabel('Active Power Demand P_{d7} (MW)');
ylabel('Reactive Power Demand Q_{d7} (Mvar)');
legend('Feasibility Boundary (Last Converged Load)', 'Base Case Load', 'Location',
'best');
axis equal; % use equal scaling
axis tight; % adjust limits to fit data
```



The power flow feasibility region for Bus 7 was determined using a continuation power flow approach. Starting from the base case load (Pd7, Qd7), the load was increased incrementally along rays of constant power factor angle. For each increment, the full Newton-Raphson power flow was solved. The boundary point for each ray was identified as the last load value for which the NRPF converged within the specified tolerance and maximum iterations, and produced physically realistic voltages (V > 0.1 p.u.). Repeating this process for multiple power factor angles (ranging from 0.85 lagging to 0.85 leading) allowed tracing the boundary curve.

The resulting plot shows the characteristic "nose curve" shape, enclosing the set of (Pd7, Qd7) combinations for which a stable power flow solution exists. The boundary represents the maximum loadability limit. The base case load point lies comfortably within this region. The shape reflects the system's ability to deliver power. The maximum active power deliverable, or the "tip" of the nose, occurs near unity power factor, while the ability to supply active power decreases as the reactive power demand (either lagging or leading) increases significantly. This boundary is often dictated by voltage stability limits.

The size of the feasibility region indicates the system's robustness to load increases at Bus 7. To enlarge this region (increase maximum loadability) we can:

1. Enhance Local Voltage Support: Add reactive power sources (e.g., shunt capacitors, STATCOMs) at or electrically near Bus 7. This directly counteracts voltage drop under heavy load, pushing the boundary outwards.

2. Reinforce Transmission: Reduce the impedance between Bus 7 and generation sources by adding parallel lines or upgrading existing ones. Less impedance naturally would reduce voltage dropr.

3. Increase Generation Capacity: Increasing generation limits could expand the feasible region, although this is often harder than addidng local VAR support.

Question 7: N-1 Security Analysis

```
contingencies = [4 9; 8 9; 4 5; 5 6; 6 7; 7 8]; % lines specified for removal
num_contingencies = size(contingencies, 1);
security_results = table('Size', [num_contingencies 4], ...
    'VariableTypes', {'string', 'double', 'double', 'string'}, ...
    'VariableNames', {'Contingency', 'Min_V_pu', 'Min_V_Bus', 'Status'});

fprintf('Running initial N-1 analysis...\n');
```

```
Running initial N-1 analysis...
```

```
for i = 1:num_contingencies
    line_out = contingencies(i, :);
    contingency_str = sprintf('Loss of Line %d-%d', line_out(1), line_out(2));
    fprintf('  Simulating: %s...\n', contingency_str);
    % find index of the line to remove
    line_idx = find( (nfrom == line_out(1) & nto == line_out(2)) | (nfrom ==
line_out(2) & nto == line_out(1)), 1 );

    if isempty(line_idx)
        warning('Contingency line %d-%d not found in branch data.', line_out(1),
line_out(2));
        security_results.Contingency(i) = contingency_str + " (Line Not Found)";
        security_results.Status(i) = "ERROR";
        continue;
    end

    % create temporary branch data excluding the contingency line
    temp_mask = true(size(nfrom)); temp_mask(line_idx) = false;
    temp_nfrom = nfrom(temp_mask); temp_nto = nto(temp_mask);
    temp_r = r(temp_mask); temp_x = x(temp_mask); temp_b = b(temp_mask);

    % build Ybus for the contingency case
    Y_contingency = admittance(temp_nfrom, temp_nto, temp_r, temp_x, temp_b);

    % NRPF for contingency
    [V_cont, ~, ~, ~, N_cont, ~] = nrpf(Y_contingency, is, ipq, ipv, Pg, Qg, Pd,
Qd, V0_full, Sbase, toler, maxiter);

    % results
    security_results.Contingency(i) = contingency_str;
    if N_cont >= maxiter || any(isnan(V_cont))
        % handle non-convergence cases
        security_results.Min_V_pu(i) = NaN;
        security_results.Min_V_Bus(i) = NaN;
        security_results.Status(i) = "NO_CONVERGENCE";
        fprintf('    -> No convergence for %s\n', contingency_str);
    else
        [min_v, min_bus] = min(V_cont);
        security_results.Min_V_pu(i) = min_v;
        security_results.Min_V_Bus(i) = min_bus;
        if min_v < 0.95
            security_results.Status(i) = "VIOLATION";
            fprintf('    -> VIOLATION: Min V = %.4f pu at Bus %d\n', min_v,
min_bus);
        else
            security_results.Status(i) = "OK";
            fprintf('    -> OK: Min V = %.4f pu at Bus %d\n', min_v, min_bus);
        end
    end
end
```

```
  Simulating: Loss of Line 4-9...
    -> VIOLATION: Min V = 0.7940 pu at Bus 9
  Simulating: Loss of Line 8-9...
    -> VIOLATION: Min V = 0.8905 pu at Bus 9
  Simulating: Loss of Line 4-5...
    -> VIOLATION: Min V = 0.9076 pu at Bus 5
  Simulating: Loss of Line 5-6...
    -> VIOLATION: Min V = 0.9190 pu at Bus 5
  Simulating: Loss of Line 6-7...
    -> VIOLATION: Min V = 0.9465 pu at Bus 7
  Simulating: Loss of Line 7-8...
    -> VIOLATION: Min V = 0.9333 pu at Bus 9
```

```
fprintf('\nInitial N-1 Security Results Summary:\n');
```

```
Initial N-1 Security Results Summary:
```

```
disp(security_results);
```

```
        Contingency         Min_V_pu    Min_V_Bus      Status
    ___________________    ________    _________    ___________

    "Loss of Line 4-9"     0.79401         9        "VIOLATION"
    "Loss of Line 8-9"     0.89047         9        "VIOLATION"
    "Loss of Line 4-5"     0.90758         5        "VIOLATION"
    "Loss of Line 5-6"     0.91898         5        "VIOLATION"
    "Loss of Line 6-7"     0.94647         7        "VIOLATION"
    "Loss of Line 7-8"     0.93331         9        "VIOLATION"
```

```
% MITIGATIOn
% first capacitor (at Bus 9)
mitigation_bus_1 = 9;
capacitor_mvar_1 = 60; % Mvar
capactiro_admittance_1 = 1j * capacitor_mvar_1 / Sbase; % calculate admittance in
p.u.

% second capacitor (at Bus 5)
mitigation_bus_2 = 5;
capacitor_mvar_2 = 60; % Mvar (adjusted to this)
capacaitor_admittance_2 = 1j * capacitor_mvar_2 / Sbase;

fprintf('Proposed Mitigation: Add %.1f Mvar shunt capacitor at Bus %d AND %.1f Mvar
at Bus %d.\n', ...
        capacitor_mvar_1, mitigation_bus_1, capacitor_mvar_2, mitigation_bus_2);
```

```
Proposed Mitigation: Add 60.0 Mvar shunt capacitor at Bus 9 AND 60.0 Mvar at Bus 5.
```

```
fprintf('Verifying mitigation effectiveness for all contingencies...\n');
```

```
Verifying mitigation effectiveness for all contingencies...
```

```

% check
mitigation_results = table('Size', [num_contingencies 3], ...
    'VariableTypes', {'string', 'double', 'string'}, ...
    'VariableNames', {'Contingency', 'Min_V_pu_Mitigated', 'Status'});

for i = 1:num_contingencies
    line_out = contingencies(i, :);
    contingency_str = sprintf('Loss of Line %d-%d', line_out(1), line_out(2));
    fprintf('  Re-simulating: %s with mitigation...\n', contingency_str);
    % find index of the line to remove
    line_idx = find( (nfrom == line_out(1) & nto == line_out(2)) | (nfrom ==
line_out(2) & nto == line_out(1)), 1 );

    if isempty(line_idx)
        mitigation_results.Contingency(i) = contingency_str + " (Line Not Found)";
        mitigation_results.Status(i) = "ERROR";
        continue;
    end

    % create temporary branch data excluding the contingency line
    temp_mask = true(size(nfrom)); temp_mask(line_idx) = false;
    temp_nfrom = nfrom(temp_mask); temp_nto = nto(temp_mask);
    temp_r = r(temp_mask); temp_x = x(temp_mask); temp_b = b(temp_mask);

    % build Ybus for the contingency case
    Y_contingency = admittance(temp_nfrom, temp_nto, temp_r, temp_x, temp_b);

    % apply both mitigation capacitors to the diagonal elements
    Y_mitigated = Y_contingency;
    if n >= mitigation_bus_1 % Ensure bus exists
       Y_mitigated(mitigation_bus_1, mitigation_bus_1) =
Y_mitigated(mitigation_bus_1, mitigation_bus_1) + capactiro_admittance_1;
    else
        warning('Mitigation bus %d does not exist in contingency network for %s?',
mitigation_bus_1, contingency_str);
    end
     if n >= mitigation_bus_2 % Ensure bus exists
        Y_mitigated(mitigation_bus_2, mitigation_bus_2) =
Y_mitigated(mitigation_bus_2, mitigation_bus_2) + capacaitor_admittance_2;
    else
        warning('Mitigation bus %d does not exist in contingency network for %s?',
mitigation_bus_2, contingency_str);
    end

    % NRPF for mitigated contingency
    [V_mit, ~, ~, ~, N_mit, ~] = nrpf(Y_mitigated, is, ipq, ipv, Pg, Qg, Pd, Qd,
V0_full, Sbase, toler, maxiter);

    % results
    mitigation_results.Contingency(i) = contingency_str;
    if N_mit >= maxiter || any(isnan(V_mit)) % Check for non-convergence or NaN
        mitigation_results.Min_V_pu_Mitigated(i) = NaN;
        mitigation_results.Status(i) = "NO_CONVERGENCE";
        fprintf('    -> NO CONVERGENCE for %s\n', contingency_str);
    else
        min_v_mit = min(V_mit);
        mitigation_results.Min_V_pu_Mitigated(i) = min_v_mit;
        if min_v_mit >= 0.95
            mitigation_results.Status(i) = "SECURE";
            fprintf('    -> SECURE: Min V = %.4f pu\n', min_v_mit);
        else
            mitigation_results.Status(i) = "INSUFFICIENT";
             fprintf('    -> STILL INSUFFICIENT: Min V = %.4f pu\n', min_v_mit); %
Added feedback
        end
    end
end
```

```
  Re-simulating: Loss of Line 4-9 with mitigation...
    -> SECURE: Min V = 0.9726 pu
  Re-simulating: Loss of Line 8-9 with mitigation...
    -> SECURE: Min V = 0.9827 pu
  Re-simulating: Loss of Line 4-5 with mitigation...
    -> SECURE: Min V = 1.0000 pu
  Re-simulating: Loss of Line 5-6 with mitigation...
    -> SECURE: Min V = 0.9902 pu
  Re-simulating: Loss of Line 6-7 with mitigation...
    -> SECURE: Min V = 0.9687 pu
  Re-simulating: Loss of Line 7-8 with mitigation...
    -> SECURE: Min V = 0.9587 pu
```

```
fprintf('\nVerification of Mitigation for All Contingencies Summary:\n');
```

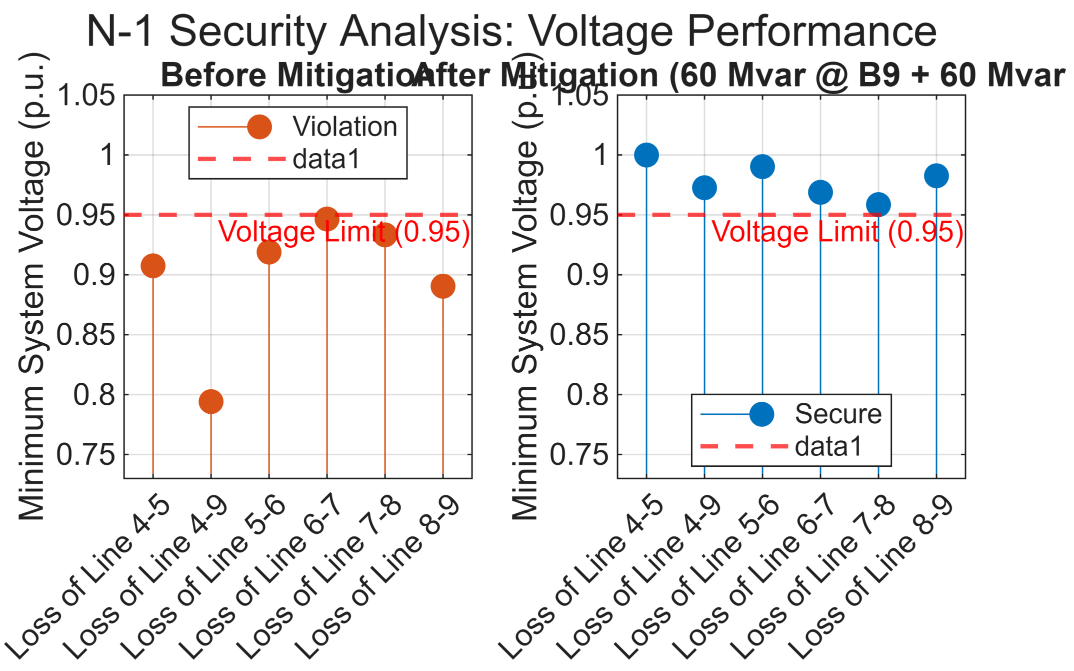
```
Verification of Mitigation for All Contingencies Summary:
```

```
disp(mitigation_results);
```

```
       Contingency          Min_V_pu_Mitigated      Status
    __________________    __________________    ________

    "Loss of Line 4-9"          0.97257           "SECURE"
    "Loss of Line 8-9"           0.9827           "SECURE"
    "Loss of Line 4-5"                1           "SECURE"
    "Loss of Line 5-6"          0.99017           "SECURE"
    "Loss of Line 6-7"          0.96874           "SECURE"
    "Loss of Line 7-8"          0.95868           "SECURE"
```

```

figure('Name', 'Q7: N-1 Security Analysis Results');
contingency_labels = categorical(security_results.Contingency); % Use initial
results for labels
valid_idx_pre = ~isnan(security_results.Min_V_pu); % Index for valid pre-mitigation
results
valid_idx_post = ~isnan(mitigation_results.Min_V_pu_Mitigated); % Index for valid
post-mitigation results

% Before Mitigation
subplot(1, 2, 1);
violation_idx_pre = valid_idx_pre & (security_results.Min_V_pu < 0.95); % Find
violations

% Plot OK points (if any exist pre-mitigation)
if any(valid_idx_pre & ~violation_idx_pre)
    stem(contingency_labels(valid_idx_pre & ~violation_idx_pre), ...
         security_results.Min_V_pu(valid_idx_pre & ~violation_idx_pre), ...
         'filled', 'Color', '#0072BD', 'MarkerSize', 8, 'DisplayName', 'OK'); %
Blue for OK
    hold on;
end

% Plot VIOLATION points
if any(violation_idx_pre)
    stem(contingency_labels(violation_idx_pre), ...
         security_results.Min_V_pu(violation_idx_pre), ...
         'filled', 'Color', '#D95319', 'MarkerSize', 8, 'DisplayName',
'Violation'); % Red for Violation
    if ~any(valid_idx_pre & ~violation_idx_pre) % If hold wasn't turned on yet
        hold on;
    end
end

yline(0.95, '--', 'Voltage Limit (0.95)', 'Color', 'r', 'LineWidth', 1.5, ...
      'LabelVerticalAlignment', 'bottom');
grid on; box on;
```

```
% Determine combined Y limits for consistent scaling
all_min_voltages = [security_results.Min_V_pu(valid_idx_pre);
mitigation_results.Min_V_pu_Mitigated(valid_idx_post)];
ylim_min = min(0.75, min(all_min_voltages(~isnan(all_min_voltages)))) - 0.02;
ylim_max = 1.05; % Keep upper limit fixed at 1.05
ylim([ylim_min ylim_max]);
title('Before Mitigation');
ylabel('Minimum System Voltage (p.u.)');
xtickangle(45);
legend('show', 'Location', 'best');

% After Mitigation
subplot(1, 2, 2);
viol_idx_post = valid_idx_post & (mitigation_results.Min_V_pu_Mitigated < 0.95); %
Find remaining violations

% Plot SECURE points
stem(contingency_labels(valid_idx_post & ~viol_idx_post), ...
     mitigation_results.Min_V_pu_Mitigated(valid_idx_post & ~viol_idx_post), ...
     'filled', 'Color', '#0072BD', 'MarkerSize', 8, 'DisplayName', 'Secure'); %
Blue for Secure
hold on;

% Plot INSUFFICIENT points (if any remain)
if any(viol_idx_post)
    stem(contingency_labels(viol_idx_post), ...
         mitigation_results.Min_V_pu_Mitigated(viol_idx_post), ...
         'filled', 'Color', '#EDB120', 'MarkerSize', 8, 'DisplayName',
'Insufficient'); % Yellow for Insufficient
end

yline(0.95, '--', 'Voltage Limit (0.95)', 'Color', 'r', 'LineWidth', 1.5, ...
      'LabelVerticalAlignment', 'bottom');
grid on; box on;
ylim([ylim_min ylim_max]); % Use same y-limits as left plot for direct comparison
title(sprintf('After Mitigation (%.0f Mvar @ B%d + %.0f Mvar @ B%d)', ...
        capacitor_mvar_1, mitigation_bus_1, capacitor_mvar_2, mitigation_bus_2)); %
Updated title
ylabel('Minimum System Voltage (p.u.)');
xtickangle(45);
sgtitle('N-1 Security Analysis: Voltage Performance'); % Overall figure title
legend('show', 'Location', 'best');
```



```
Plotting complete.
```

N-1 security analysis was performed on the IEEE 9-bus system using the previously validated full Newton-Raphson power flow function to assess system performance under contingencies. The initial simulations indicated that the loss of any specified line resulted in minimum system voltages falling below the 0.95 p.u. operational limit. Voltage violations were most severe at buses electrically distant from generation sources, specifically Buses 9 and 5, confirming the principle that voltage stability requires adequate local reactive power support.

Based on these findings, a mitigation strategy employing shunt capacitor banks was adopted. Given Bus 9's vulnerability, an initial capacitor was placed there. To address remaining voltage issues primarily affecting Bus 5 during specific outages (lines 4-5 and 5-6), a second capacitor was placed at Bus 5. After trial and error, we decided to place capacitors of 60 MVARs at both nodes.

The N-1 analysis was repeated with both capacitors added into the network model for each contingency case. The results confirmed that the local reactive loads were sufficient in mitigating the line losses, with the minimum voltage remaining above the 0.95 p.u. threshold for all tested contingencies.

Conclusion

This assignment successfully implemented and validated four distinct power flow solution methods: full Newton-Raphson (NRPF), Decoupled (DECPF), Fast-Decoupled (FDPF), and the linear DC Power Flow (DCPF). The algorithms were applied to the IEEE 9-bus system, allowing for a thorough comparison of their numerical performance, accuracy, and adherence to physical power system principles.

Across all AC methods, convergence was achieved within the specified tolerance, with the NRPF exhibiting the fastest convergence rate (3 iterations) compared to the linearly convergent DECPF and FDPF (6 iterations), as theoretically expected. Numerical checks confirmed the accuracy of the implementations, with DECPF and FDPF producing voltage and angle results nearly identical to the NRPF benchmark, validating the physical P-delta and Q-V decoupling assumption for this system. Computational timings highlighted the efficiency gains of the decoupled methods, particularly the FDPF, which leverages pre-factorized constant matrices. The DCPF provided a rapid, non-iterative estimate of active power flows and angles, suitable for preliminary studies.

The DCPF results demonstrated its inherent limitation caused by neglecting losses, as highlighted by the lower calculated slack power compared to the AC methods.. The comparison between AC active power (P) and apparent power (S) flows reinforced that DCPF, while useful for P estimation, is insufficient for thermal limit assessments which depend on S.

Finally, the practical application of these tools was demonstrated. The power flow feasibility region for load increases at Bus 7 was successfully mapped using a continuation NRPF approach, illustrating voltage stability limits. An N-1 security analysis, also using NRPF, identified contingencies causing voltage violations. A mitigation strategy based on local reactive power compensation (shunt capacitors) was designed and verified, successfully restoring acceptable voltage profiles across all contingencies. This confirmed the physical principle that maintaining voltage stability requires local reactive power sources.

Overall, this assignment demonstrated the successful implementation and application of several power flow analysis methods, confirming both the numerical robustness of the algorithms and their ability to reflect the physical behavior of power systems under various operating conditions and contingencies.

Appendix

Link to repository containing all functions: https://github.com/dtemurcu/DT_A2_temp