% Add the bimat package to the path

% for work laptop

%addpath(genpath('/Users/emilypayne/Documents/bimat\_packadge'));

% for epg desktop

addpath(genpath('C:\Users\emp.payne\Documents\bimat\_github\_repo'));

% Read the bipartite matrix from the Excel file

matrix = readmatrix('9bus bipartite matrices.xlsx', 'Sheet', 'WSCC9 Grid Original Matrix All', 'Range', 'B2:P43');

% Initialize variables

[m, n] = size(matrix);

outputData = []; % Initialize outputData to store results

% Define max connections for specific row and column ranges

maxConnectionsRowRange1 = 4; % Max 4 connections for rows 1-21

maxConnectionsRowRange2 = 9; % Max 9 connections for rows 28-42

maxConnectionsRow40to42 = 6; % Max 6 connections for rows 40-42

maxConnectionsColumns1to9 = 10; % Max 10 connections for columns 1-9

maxConnectionsColumns13to15 = 8; % Max 8 connections for columns 13-15

% Loop to add all possible connections one by one

while true

bestNODF = -inf;

bestPosition = [];

% Find the best position to add a new connection

for i = 1:m

for j = 1:n

if matrix(i, j) == 0

% Determine the max connections allowed for the current row

if i >= 1 && i <= 21

maxConnectionsPerRow = maxConnectionsRowRange1; % Max 4 for rows 1-21

elseif i >= 28 && i <= 39

maxConnectionsPerRow = maxConnectionsRowRange2; % Max 9 for rows 28-39

elseif i >= 40 && i <= 42

maxConnectionsPerRow = maxConnectionsRow40to42; % Max 6 for rows 40-42

else

maxConnectionsPerRow = inf; % No limit for other rows

end

% Determine the max connections allowed for the current column

if j >= 1 && j <= 10

maxConnectionsPerColumn = maxConnectionsColumns1to9; % Max 10 for columns 1-9

elseif j >= 13 && j <= 16

maxConnectionsPerColumn = maxConnectionsColumns13to15; % Max 8 for columns 13-15

else

maxConnectionsPerColumn = inf; % No limit for other columns

end

% Check row and column constraints before adding the connection

if sum(matrix(i, :)) < maxConnectionsPerRow && sum(matrix(:, j)) < maxConnectionsPerColumn

matrix(i, j) = 1; % Temporarily add the connection

[currentNODF, ~, ~] = calculate\_NODF(matrix); % Calculate NODF

if currentNODF > bestNODF % Update if better NODF is found

bestNODF = currentNODF;

bestPosition = [i, j];

end

matrix(i, j) = 0; % Revert the matrix to its original state

end

end

end

end

% Break loop if no more connections can be added

if isempty(bestPosition)

break;

end

% Add the best connection found

matrix(bestPosition(1), bestPosition(2)) = 1;

% Compute Qb, Qr, NODF, and Num\_mod

[Qb, Qr, NODF\_N, NODF\_cols, NODF\_rows, Num\_mod] = computeQbAndMetrics(matrix);

% Save the matrix for this step

matrixFilename = ['9bus\_normal\_optim\_lim\_matrix\_step\_', num2str(size(outputData, 1) + 1), '.xlsx'];

writematrix(matrix, matrixFilename);

disp(['Matrix saved to ', matrixFilename]);

% Save the result for this step, including Num\_mod

outputData = [outputData; bestPosition(1), bestPosition(2), Qb, Qr, NODF\_N, bestNODF, NODF\_cols, NODF\_rows, Num\_mod];

disp(['New connection added at position: (', num2str(bestPosition(1)), ', ', num2str(bestPosition(2)), ')']);

disp(['Qb: ', num2str(Qb), ', Qr: ', num2str(Qr), ', N: ', num2str(NODF\_N), ...

', NODF\_cols: ', num2str(NODF\_cols), ', NODF\_rows: ', num2str(NODF\_rows), ', Num\_mod: ', num2str(Num\_mod)]);

end

% Combine headers and data into a cell array

headers = {'Row', 'Column', 'Qb', 'Qr', 'NODF\_N', 'bestNODF', 'NODF\_cols', 'NODF\_rows', 'Num\_mod'};

combinedData = [headers; num2cell(outputData)];

% Save the combined data to an Excel file

resultsFilename = '9bus\_normal\_optim\_lim\_connection\_results.xlsx';

writecell(combinedData, resultsFilename);

disp(['Results saved to ', resultsFilename]);

% Function to calculate NODF and related metrics

function [NODF\_N, NODF\_cols, NODF\_rows] = calculate\_NODF(matrix)

bp = Bipartite(matrix);

nest\_set = bp.nestedness.Detect();

NODF\_N = nest\_set.N;

NODF\_cols = nest\_set.N\_cols;

NODF\_rows = nest\_set.N\_rows;

end

% Function to compute Qb, Qr, NODF, and Num\_mod

function [Qb, Qr, NODF\_N, NODF\_cols, NODF\_rows, Num\_mod] = computeQbAndMetrics(matrix)

bp = Bipartite(matrix);

mod\_set = bp.community.Detect();

Num\_mod = mod\_set.N; % Assign Num\_mod

Qb = mod\_set.Qb;

Qr = mod\_set.Qr;

[NODF\_N, NODF\_cols, NODF\_rows] = calculate\_NODF(matrix);

end