%% 8-point Radix-2 FFT for all rows and plot on a single graph with specified legend entries

clear all;

clc;

close all;

% Rectangular antenna/element array for metasurface

% Parameters

Nx = 4; % Number of elements in the x-dimension

Ny = 4; % Number of elements in the y-dimension

d = 0.5; % Distance between elements in wavelengths

lambda = 1; % Wavelength

k = 2 \* pi / lambda; % Wave number

fs = 1e3; % Sampling frequency

t = 0:1/fs:1; % Time vector

A = 1; % Amplitude of the incoming signal

% Create grid of element positions

[x, y] = meshgrid(0:Nx-1, 0:Ny-1);

x = x \* d;

y = y \* d;

% Generate the incoming signal (a simple sinusoidal wave in this example)

signal = A \* cos(2 \* pi \* 100 \* t); % Signal as a cosine wave at 100 Hz

% Initialize the received signal matrix

received\_signal\_data = []; % Store received signal data

% Total number of iterations (for progress display)

total\_iterations = length(-5:2:5) \* length(-90:5:90) \* length(-90:5:90);

current\_iteration = 0;

% Loop over different SNR, theta, phi values

for SNR = -5:2:5 % Reduce the step size for theta and phi

for theta = -90:5:90 % Increased step size

for phi = -90:5:90 % Increased step size

% Update progress display

current\_iteration = current\_iteration + 1;

disp(['Processing: ', num2str(current\_iteration), ' of ', num2str(total\_iterations)]);

% Calculate the received signal for this combination of parameters

received\_signal = calculate\_signal(SNR, theta, phi, Nx, Ny, x, y, signal, k, t);

% Store the received signal data

received\_signal\_data = [received\_signal\_data; SNR, theta, phi, received\_signal];

end

end

end

% Function to calculate the signal based on SNR, theta, and phi

function signal\_out = calculate\_signal(SNR, theta, phi, Nx, Ny, x, y, signal, k, t)

% Calculate Array Factor

Array\_factor = ArrayFactor(theta, phi, Nx, Ny, x, y, k);

% Ensure Array\_factor is a scalar for multiplication with the signal

% Signal is a time-domain signal, so multiply it with the scalar Array\_factor

received\_signal = Array\_factor \* signal; % Multiply by signal

% Adjust received signal by the SNR (simple model)

noise = (1 / 10^(SNR / 10)) \* randn(size(received\_signal)); % Add noise based on SNR

signal\_out = received\_signal + noise; % Add noise to the signal

end

% Function to calculate the Array Factor

function Array\_factor = ArrayFactor(theta, phi, Nx, Ny, x, y, k)

% Initialize Array Factor

Array\_factor = 0;

% Calculate the array factor based on the positions of the elements

for i = 1:Nx

for j = 1:Ny

% Calculate the phase shift for element (i,j) due to its position

phase\_shift = k \* (x(i,j) \* sin(deg2rad(theta)) \* cos(deg2rad(phi)) + ...

y(i,j) \* sin(deg2rad(theta)) \* sin(deg2rad(phi)));

% Add the contribution from this element to the array factor

Array\_factor = Array\_factor + exp(1j \* phase\_shift);

end

end

% Normalize the Array Factor (optional, if needed)

% Array\_factor = Array\_factor / (Nx \* Ny); % Uncomment if you want to normalize

end

% Initialize table to store FFT values, theta, and phi

fft\_table\_data = [];

% Process each row of received signal data

for row = 1:size(received\_signal\_data, 1)

% Extract FFT data for the current row (first 8 FFT values)

fft\_values = abs(fft(received\_signal\_data(row, 4:11), 8));

% Normalize the FFT values to the range [0, 1]

normalized\_fft\_values = (fft\_values - min(fft\_values)) / (max(fft\_values) - min(fft\_values));

% Extract theta and phi values

theta = received\_signal\_data(row, 2); % Theta value

phi = received\_signal\_data(row, 3); % Phi value

% Combine the normalized FFT values with theta and phi

fft\_table\_data = [fft\_table\_data; normalized\_fft\_values, theta, phi];

end

% Create a table for the data

fft\_table = array2table(fft\_table\_data, ...

'VariableNames', {'FFT\_1', 'FFT\_2', 'FFT\_3', 'FFT\_4', ...

'FFT\_5', 'FFT\_6', 'FFT\_7', 'FFT\_8', ...

'Theta', 'Phi'});

% Display the first 10 rows of the table for verification

disp(fft\_table(1:10, :)); % Show the first 10 rows

% Optionally, export the table to a CSV file for further use

writetable(fft\_table, 'FFT\_Normalized\_Table.csv');

disp('Table saved as "FFT\_Normalized\_Table.csv".');

%% Data Preparation

% Randomize rows in a non-uniform fashion

rng(42); % Set random seed for reproducibility

randomized\_indices = randperm(size(fft\_table\_data, 1));

fft\_table\_data = fft\_table\_data(randomized\_indices, :);

% Split data into inputs (FFT values) and outputs (Theta and Phi)

inputs = fft\_table\_data(:, 1:8); % FFT values

outputs = fft\_table\_data(:, 9:10); % Theta and Phi

% Normalize Theta and Phi to range [0, 1] for ANN training

outputs = (outputs - min(outputs, [], 1)) ./ (max(outputs, [], 1) - min(outputs, [], 1));

% Split data into training and testing sets (80% for training)

split\_ratio = 0.8;

num\_training\_samples = floor(split\_ratio \* size(inputs, 1));

training\_inputs = inputs(1:num\_training\_samples, :);

training\_outputs = outputs(1:num\_training\_samples, :);

testing\_inputs = inputs(num\_training\_samples+1:end, :);

testing\_outputs = outputs(num\_training\_samples+1:end, :);

%% Create and Train the Recurrent Neural Network

% Define an RNN with 300 hidden neurons and feedback connections

layerDelays = 1:2; % Delays for the input layer

hiddenSizes = 10; % Number of hidden neurons

trainFcn = 'trainlm';

% Create a layer recurrent network

net = layrecnet(layerDelays,hiddenSizes,trainFcn);

% Configure network training options

net.trainFcn = 'trainlm'; % Use Levenberg-Marquardt algorithm

net.trainParam.epochs = 500; % Maximum number of epochs

net.trainParam.goal = 1e-5; % Training goal

net.trainParam.showCommandLine = true; % Display training progress in the command line

net.trainParam.showWindow = false; % Disable GUI window

% Prepare the data as sequences

training\_inputs\_seq = con2seq(training\_inputs'); % Convert training inputs to sequences

training\_outputs\_seq = con2seq(training\_outputs'); % Convert training outputs to sequences

% Train the RNN

[net, tr] = train(net, training\_inputs\_seq, training\_outputs\_seq);

%% Evaluate the Recurrent Neural Network

% Prepare testing data as sequences

testing\_inputs\_seq = con2seq(testing\_inputs'); % Convert testing inputs to sequences

% Predict outputs using the trained RNN

predicted\_outputs\_seq = net(testing\_inputs\_seq);

% Convert predicted outputs back to matrix form

predicted\_outputs = cell2mat(predicted\_outputs\_seq)';

% Calculate performance metrics

mse\_error = mean((predicted\_outputs - testing\_outputs).^2, 'all');

mae\_error = mean(abs(predicted\_outputs - testing\_outputs), 'all');

% Calculate R-squared (coefficient of determination)

ss\_res = sum((testing\_outputs - predicted\_outputs).^2, 'all');

ss\_tot = sum((testing\_outputs - mean(testing\_outputs, 'all')).^2, 'all');

r\_squared = 1 - (ss\_res / ss\_tot);

% Display performance metrics

disp('Performance Metrics:');

disp(['Mean Squared Error (MSE): ', num2str(mse\_error)]);

disp(['Mean Absolute Error (MAE): ', num2str(mae\_error)]);

disp(['R-squared (R^2): ', num2str(r\_squared)]);

%% Save the Network and Results

% Save the trained RNN for later use

save('trained\_RNN.mat', 'net');

% Save the predictions and actual values for further analysis

results\_table = array2table([testing\_outputs, predicted\_outputs], ...

'VariableNames', {'Theta\_Actual', 'Phi\_Actual', 'Theta\_Predicted', 'Phi\_Predicted'});

writetable(results\_table, 'RNN\_Results.csv');

disp('Results saved as "RNN\_Results.csv".');

%% Visualize Performance

% Plot actual vs predicted values for Theta

figure;

subplot(2, 1, 1);

plot(testing\_outputs(:, 1), 'b-', 'DisplayName', 'Actual Theta'); % Actual in blue

hold on;

plot(predicted\_outputs(:, 1), 'r--', 'DisplayName', 'Predicted Theta'); % Predicted in red

xlabel('Sample Index');

ylabel('Theta');

title('Actual vs Predicted Theta');

legend('Location', 'best');

grid on;

hold off;

% Plot actual vs predicted values for Phi

subplot(2, 1, 2);

plot(testing\_outputs(:, 2), 'b-', 'DisplayName', 'Actual Phi'); % Actual in blue

hold on;

plot(predicted\_outputs(:, 2), 'r--', 'DisplayName', 'Predicted Phi'); % Predicted in red

xlabel('Sample Index');

ylabel('Phi');

title('Actual vs Predicted Phi');

legend('Location', 'best');

grid on;

hold off;

% Display residuals

figure;

residuals\_theta = testing\_outputs(:, 1) - predicted\_outputs(:, 1);

residuals\_phi = testing\_outputs(:, 2) - predicted\_outputs(:, 2);

subplot(2, 1, 1);

plot(residuals\_theta, 'g.', 'DisplayName', 'Residuals'); % Residuals in green

title('Residuals for Theta');

xlabel('Sample Index');

ylabel('Residual');

legend('Location', 'best');

grid on;

subplot(2, 1, 2);

plot(residuals\_phi, 'g.', 'DisplayName', 'Residuals'); % Residuals in green

title('Residuals for Phi');

xlabel('Sample Index');

ylabel('Residual');

legend('Location', 'best');

grid on;